521-S/W-058

PCI10FR Windows NT Device Driver Software Definition Document

March 1997

Prepared by:

Miles T. Smith, Electrical Engineer   Date
Microelectronic Systems Branch
Code 521, NASA/GSFC

Approved by:

Barbara E. Brown, Computer Engineer   Date
Microelectronic Systems Branch
Code 521, NASA/GSFC

Approved by:

Nicholas J. Speciale, Head   Date
Microelectronic Systems Branch
Code 521, NASA/GSFC

Goddard Space Flight Center
Greenbelt, Maryland

Preface

This document describes the software design and implementation of the Windows NT PCI10FR Device Driver and PCI10FR Wrapper Application Program Interface (API).

Requests for copies of this document, along with questions or proposed changes, should be addressed to:

L. Kane, Sr. Documentation Specialist [(301) 286-8609 or (301) 286-1768 FAX]
Microelectronic Systems Branch, Code 520.9
Goddard Space Flight Center
Greenbelt, Maryland 20771

Change Information Page

List of Effective Pages	
Page Number	**Issue**
Title	Draft
Signature Page	Draft
iii through viii	Draft
1-1 through 1-7	Draft
2-1 through 2-6	Draft
3-1 through 3-25	Draft
4-1 through 4-7	Draft
5-1 through 5-6	Draft
6-1 and 6-2	Draft
	Draft

Document History			
Document Number	**Status/Issue**	**Publication Date**	**CCR Number**

521-S/W-058	Draft	March 1997	

DCN Control Sheet

DCN Number	Date/Time Group (Teletype Only)	Month/Year	Section(s) Affected	Initials

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT'D)

TABLE OF CONTENTS (CONT'D)

FIGURES

TABLES

Section 1 . General Information

1.1 General Description

1.1.1 Purpose of Document

This document describes the requirements, design and implementation of the Microsoft Windows NT 3.51 kernel-mode device driver and wrapper functions for the Peripheral Component Interconnect (PCI) 10 Mbps Frame Synchronization Reed-Solomon Error Detection and Correction board, also known as the PCI10FR Prototype Board.

The PCI10FR Prototype Board was developed by the Mission Operations and Data Systems Directorate Microelectronic Systems Branch, Code 521, at Goddard Space Flight Center (GSFC). The project was funded by the System Operation and Management Office (SOMO).

1.1.2 Scope of Document

This Software Definition Document (SDD) primary focuses on the design and implementation of the PCI10FR Device Driver and the PCI10FR Wrapper Application Program Interface (API) programs. A complete hardware description of the PCI10FR Prototype Board is not provided in this document, but this information is available in the *PCI10FR Hardware Definition Document* listed in Section 1.5. The SDD describes the PCI10FR Prototype Board hardware only to the extent necessary to understand the interface between the driver and board. Similarly, information about the PCI10FR Debugger application software is described in this document only to the extent necessary to understand the PCI10FR Wrapper API, which provides a consistent application interface to the device driver.

Although some background information is provided, this document is not a tutorial on the Microsoft Windows NT 3.51 operating system or on Windows NT device driver development. The Win32 Device Driver Kit (DDK) documents listed in Section 1.5 provide details on developing Windows NT device drivers.

1.1.3 Background

The Microelectronic Systems Branch (Code 521) has been designing and implementing state-of-the-art high-rate telemetry processing Very Large Scale Integration (VLSI) Application-Specific Integrated Circuits (ASIC's) and boards for many years. Most of these boards communicate over a VME bus and use the VxWorks real-time operating system kernel. In an effort to significantly reduce the size and cost of telemetry processing technology, Code 521 has developed a new telemetry processing prototype board that uses the Peripheral Component Interconnect (PCI) bus technology, requires no on-board Central Processing Unit (CPU), and operates under the Microsoft Windows NT 3.51 operating system on low-cost Personal Computer platforms, such as Pentium-Pro systems.

The new prototype board, known as the PCI10FR Prototype Board, combines all of the front-end telemetry processing functions. Specifically, the board performs the following tasks:

a. Receipt of return link telemetry data up to 10 Mbps via RS-422 bit-serial telemetry streams..

b. Frame synchronization and CRC error detection of telemetry data in a variety of formats: Time Division Multiplexed (TDM) data, Consultative Committee for Space Data Systems (CCSDS) data, and the National Oceanic and Atmospherics Administration (NOAA) and Geostationary Orbiting Earth Satellite (GOES) weather data formats

c. Reed-Solomon (RS) error detection and correction of RS-encoded data.

d. Generation of quality, timing, and accounting annotation.

e. Distribution and sorting of up to four virtual channels into separate First-In First-Out (FIFO) queues.

f. Transmission of spacecraft forward link command data up to 1 Mbps.

Getting all of these functions on one board was made possible by Code 521's recent development of a Parallel Integrated Frame Synchronizer (PIFS) ASIC, which is capable of handling data rates up to 528 Mbps. The PCI10FR Prototype Board, which is the first board to incorporate the new PIFS chip, also uses Code 521's proven Reed-Solomon Error Correction (RSEC) ASIC.

Code 521 has developed a new ASIC, called the Service Processor (SP) chip, which will provide frame and packet services for CCSDS data. This chip, along with the PIFS and RSEC chips, will be used on the next generation Return Link Processor Card (RLPC). The RLPC will be capable of handling data rates up to 50 Mbps on a PC, and up to 150 Mbps on a Dec Alpha workstation.

1.2 Primary Functions

Under Windows NT, a user-mode application cannot "talk" directly to hardware, but must communicate with the hardware through a kernel-mode driver. The primary function of the PCI10FR Wrapper API is to provide a consistent API to a user-mode application for getting PCI10FR Prototype Board status and for configuring the PCI10FR Prototype Board chips and registers. The primary functions of the PCI10FR Device Driver are to provide the PCI10FR Wrapper API with the ability to read and write to any board memory location, to perform DMA transfers of return link telemetry data from the board to the host memory, and to transfer forward link command data from the host memory to the board. These functions are discussed in detail in Section 2 of this document, and the implementation of these functions is discussed in Section 3.

1.3 PCI10FR Board and Operational Scenarios

As shown in Figure 1-1, the PCI10FR Prototype Board consists of three main subsystems: the Telemetry Input Subsystem, the Forward Link Output Subsystem, and the Board Support Subsystem. Each of these subsystems is described below. Additional details about the board and its memory map can be found in the *PCI10FR Hardware Description Document* referenced in Section 1.5.

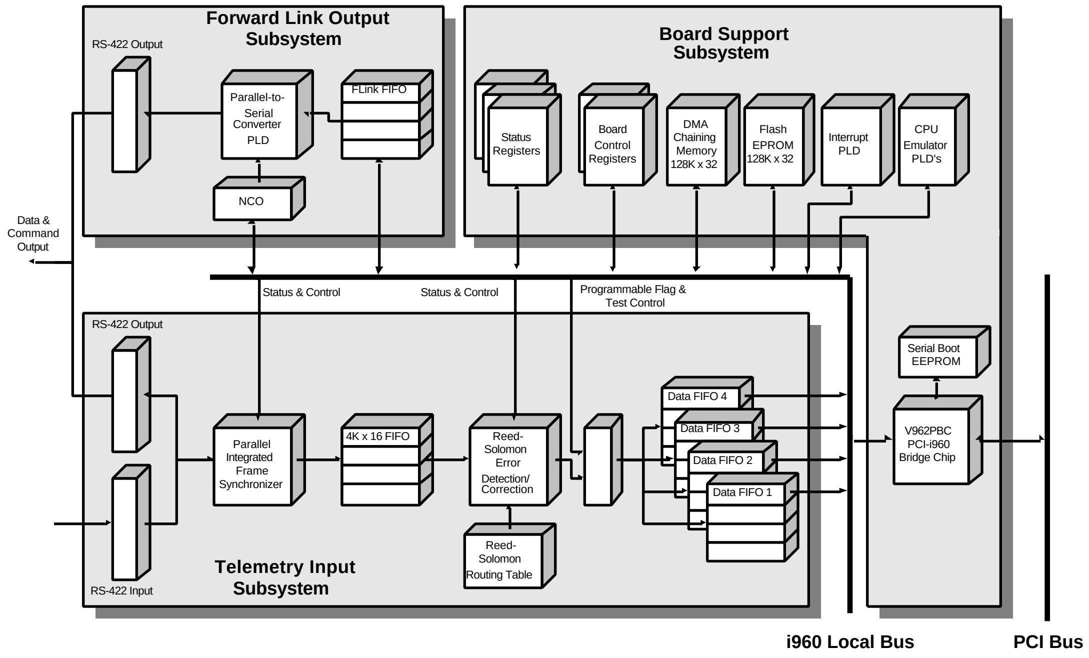


Figure 1-1. PCI10FR Prototype Board Functional Block Diagram

1.3.1 Telemetry Input Subsystem

The Telemetry Input Subsystem provides an RS-422 input port for return link telemetry data and clock signals originating from a bit synchronizer. The data and clock signals are fed into the PIFS chip which performs frame synchronization, Pseudo-Noise (PN) decoding, and CRC error detection. Quality and time stamp annotation may be optionally added to the data stream, which can then be output from the PIFS chip in either 8-bit or 16-bit words. The data is then buffered by a 4-Kbyte x 16-bit FIFO and fed into the RSEC chip.

The RSEC chip can operate in pass-through mode, to accommodate non-RS-encoded data, or it can perform Reed-Solomon error detection and correction of the data. Quality annotation may be optionally added to the data stream. The RSEC chip provides the capability of outputting the entire data stream, outputting any subset of the data stream, and moving the PIFS quality, RSEC quality, and time annotation to the either the front or rear of the data stream. The RSEC then looks at specific, user-defined contiguous 16-bit fields within the data stream, and uses the values in these fields as an index into a 64-Kbyte RS Routing Table containing information as to which of the four 32-Kbyte x 32-bit FIFO banks should receive the data stream. The RS Routing Table can route the entire data stream to any one or more of the FIFO’s, or more commonly, it can route the entire data stream to one FIFO for a composite data stream, and route specific virtual channels to the other three FIFO’s.

Four 32-Kbyte x 32-bit FIFO banks are provided on the PCI10FR Prototype Board. Each bank actually consists of four 8-Kbyte x 8-bit FIFO’s. Two Stacker programmable logic devices (PLD’s) are used to take the 8-bit output of the RSEC and stack these bytes into the four 8-bit FIFO’s. This method effectively queues the data in 32-bit words that maximize efficient DMA transfer across the 32-bit PCI bus. Each of the FIFO’s has programmable almost-empty and almost-full flags. When the almost-full flag is asserted, an interrupt is generated by the PCI10FR

Prototype Board and the PCI10FR device driver's interrupt service routine (ISR) is called. The device driver sets up a DMA operation to move data from the FIFO to host memory. The composite data stream is generally archived to disk, and the real-time data stream is transmitted over the network to the end-user. Section 3 of this document will provide design and implementation details concerning the device driver's interrupt service routine and DMA operations.

1.3.2 Forward Link Output Subsystem

Command data is received by the host computer over the network from the end-user. The device driver transmits this command data stream across the PCI bus and buffers it in a 4-Kbyte x 8-bit forward link FIFO. A parallel-to-serial (P2S) PLD is used to convert the 8-bit parallel command data stream into a serial stream, which is then output through an RS-422 port. A programmable Numerically-Controlled Oscillator (NCO) chip is used to generate the clock signal to clock the command data stream out a specific rate. An external clock signal may also be used instead of the NCO clock signal. The device driver contains routines that support forward link output, but this function of the PCI10FR Prototype Board has not been fully implemented or tested.

1.3.3 Board Support Subsystem

The Board Support Subsystem (BSS) includes several chips that are critical for the board to work, and some chips that really do not add any identified value. The most important BSS component is the V962 PCI Bridge Chip (V962PBC), produced by the V3 Semiconductor, Inc. This chip provides a bridge between the i960 local bus on the board and the PCI bus. It contains all of the important PCI configuration registers that allow the board to be recognized as a PCI device. The V962PBC chip provides registers for base memory addresses and interrupt configuration, as well as providing two DMA engines. All reads and writes to the PCI10FR Prototype Board, DMA operations, and interrupts are performed through this chip. Section 1.5 lists the reference document for this chip, as well as several references on PCI bus architecture.

When the host computer is booted, PCI configuration information is serially downloaded into the V962PBC chip through an on-board Serial Boot Electrically-Erasable Programmable Read-Only Memory (EEPROM) chip. This EEPROM chip, as well as all of the PLD's on the board, are all programmed at production time and soldered on the board. All of these chips can be reprogrammed on-board if needed.

A Temperature chip has been provided on the board to monitor the board temperature. This chip is not necessary for operation of the board.

There are three board Status Registers that provide information about the FIFO programmable flag status, and are used to read the temperature from the Temperature chip. These registers are queried during the ISR to determine which FIFO or FIFO's require servicing.

There is a Main Control Register that resets the PIFS chip, RS chip, and all of the FIFO's. The Main Control Register also sets the stacking order (big-endian or little-endian) and stacking mode (packed frames or padded frames) for the two Stacker PLD's, and it is used to program the NCO clock frequencies. This register, along with a Miscellaneous Control Register, provide for masking of specific board interrupts. The Miscellaneous Control Register is also used to send commands and programming information to the Temperature chip, and to toggle four software definable bits which connect to test points on the board. These test points have been invaluable in measuring interrupt and deferred procedure call (DPC) latencies using a logic analyzer.

A Programmable Flag/Test Data Register is provided to program the FIFO flags. The PIFS and RSEC chip registers all have their own address space and can be accessed directly, as can the four return link data FIFO's and one forward link command FIFO. The PCI10FR Prototype Board contains 128 Kbytes of Random Access Memory (RAM) that can be used for DMA chaining (scatter-gather). Unfortunately, the current revision (Revision B2) of the V962PBC chip does not support DMA chaining. The next revision, Revision C0, will support this feature.

The PCI10FR Prototype Board has a number of PLD's used for emulating a local bus CPU. These PLD's provide all of the chip select and acknowledge signals on the local bus. There is also an Interrupt PLD that allows the interrupt signals on the local bus to be masked, depending on the contents of the Main Control Register and Miscellaneous Control Register. Finally, the PCI10FR Prototype Board has 128 Kbytes of Flash EEPROM, which is not used for anything.

1.4 Document Organization

Section 1 of this document provides an overview of the PCI10FR Prototype Board and the device driver. Section 2 describes the functional requirements and context of the PCI10FR Device Driver and the PCI10FR Wrapper API. Section 3 describes how the device driver is designed to perform its tasks. Section 4 provides implementation information and details on how to build the software. Section 5 addresses development issues that were identified during the course of the project. Section 6 describes testing and debugging of the PCI10FR Device Driver. Appendix A provides a brief user's guide to the PCI10FR Debugger application. Numerous other appendices, listed in the table of contents, provide specification sheets and other internal information that may be difficult to find otherwise. Finally, a very necessary list of acronyms and abbreviations is provided at the end of this document.

1.5 Reference Documents

a. *PCI 10-Mbps Frame Synchronization Reed-Solomon Error Detection and Correction (EDC) Card Preliminary Design Review*, March 1, 1996, Code 521, NASA/GSFC.

b. *PCI 10-Mbps Frame Synchronization Reed-Solomon EDC Card Critical Design Review*, April 18, 1996, Code 521, NASA/GSFC.

c. *PCI10FR Hardware Definition Document*, Early Draft, April 25, 1996, Code 521, NASA/GSFC.

d. *Parallel Integrated Frame Synchronizer Chip*, 521-ASIC-023, December 1996, Code 521, NASA/GSFC.

e. "Reed-Solomon Error Correction Chip", Section 10, *Microelectronic Systems Branch Application-Specific Integrated Circuits (ASIC) Components Document*, Volume 1, 521-SPEC-002, Revision 1, August 1995, Code 521, NASA/GSFC.

f. *CMOS SyncFIFO IDT72241*, DSC-2655/6, December 1995, Integrated Device Technology, Inc..

g. *PCI10FR: Programming the FIFO Flags*, Email from Ken Winiecki, July 19, 1996.

h. *DS1620 Digital Thermometer and Thermostat*, DS1620, 1995, Dallas Semiconductor Corporation.

i. *Application Note 105 — High Resolution Temperature Measurement with Dallas Direct-to-Digital Temperature Sensors*, Application Note 105, 1995, Dallas Semiconductor Corporation.

j. *Dual Programmable Clock Generator*, CH9203, Rev. 1.7, February 21, 1995, Chrontel, Inc.

k. *Using CH9203 in Multiple Speed CD-ROM Drive*, AN-12 Application Notes, Rev. 1.2, February 21, 1996, Chrontel, Inc.

l. *2-Wire Serial CMOS E^2PROMs*, AT24C01A/2/4/8/16, Atmel, Inc.

m. *VxxxPBC User's Manual: Local Bus to PCI Bridge for i960, Am29K and PowerPC Processors*, Rev. 2.0, 1996, V3 Semiconductor, Inc.

n. *V292PBC, V960PBC, V961PBC, V962PBC Stepping Change Notification: 'B-0' Step to 'B-1' Step*, V3 Technical Note, Rev. 2.2, February 13, 1996, V3 Semiconductor, Inc.

o. *Stepping Change Notification: PBC 'B1' Step to 'B2' Step*, V3 Technical Note, Rev. 3.2, August 9, 1996, V3 Semiconductor, Inc.

p. *Win32 DDK: Programmer's Guide*, Microsoft Development Library, 1992-1995, Microsoft Corporation.

q. *Win32 DDK: Kernel-Mode Driver Design Guide*, Microsoft Development Library, 1992-1995, Microsoft Corporation.

r. *Win32 DDK: Kernel-Mode Driver Reference Document*, Microsoft Development Library, 1992-1995, Microsoft Corporation.

s. *Inside Windows NT*, Helen Custer, Microsoft Press, 1993.

t. *Advanced Windows: The Developer's Guide to the Win32 API for Windows NT 3.5 and Windows 95*, Jeffrey Richter, Microsoft Press, 1995.

u. *The Windows NT Device Driver Book*, Art Baker, Prentice Hall PTR, 1997.

v. *Developing Windows NT Kernel Mode Drivers*, Windows NT Device Driver Course Textbook, Open System Resources, Inc., 1996.

w. *PCI System Architecture*, Third Edition, Tom Shanley and Don Anderson, MindShare, Inc., 1995.

x. *PCI Hardware and Software Architecture and Design*, Third Edition, Edward Solari and George Willse, Annabooks, 1996.

y. *Software and Automation Systems Branch C++ Style Guide*, DSTL-96-011, Version 2.0, June 1996, Software and Automation Systems Branch (Code 522), NASA Goddard Space Flight Center.

Section 2. Functional Description

2.1 Functional Specifications/Requirements

2.1.1 Functional Requirements for the PCI10FR Wrapper API

The PCI10FR Prototype Board can run on any hardware platform supporting Version 2.1 of the PCI bus standard. Since an application that needs to communicate with the board may be running under a variety of operating systems, a decision was made to create a PCI10FR Wrapper API that would provide a well-documented and consistent PCI10FR Device Driver interface to an application, regardless of the operating system. This eliminates the need to change the application software when porting from one operating system to another. Obviously, the PCI10FR Device Driver and the interface between the PCI10FR Wrapper API and the device driver would need to be rewritten for each operating system, but at least this layered approach minimizes the amount of re-coding.

As discussed above, the main functional requirements for the PCI10FR Wrapper API include:

a. Encapsulating all operating system-specific Input/Output Control (IOCTL) calls.

b. Providing an application with a consistent, operating system-independent interface to the PCI10FR Device Driver.

2.1.2 Functional Requirements for the PCI10FR Device Driver

The main functional requirements for the PCI10FR Device Driver include:

a. Probing the PCI bus to determine if a PCI10FR Prototype Board has been installed.

b. Creating a device object for the PCI10FR Prototype Board and allocating memory and interrupt resources.

c. Resetting and initializing the board in a known safe state.

d. Providing routines to dispatch I/O control (IOCTL) calls from the user-mode application to the appropriate kernel-mode routines.

e. Providing the user-mode application with the capability to read from and write to any board memory location.

f. Providing the user-mode application with the capability to read from and write to any PCI configuration space register on the board. This capability is very useful in debugging the board, but it generally should not be made available to an operator. The values in the PCI configuration space are critical for the proper and efficient operation of the board. Writing improper values in this space will cause system crashes and board failure This version of the PCI10FR Device Driver does not implement any PCI configuration write IOCTL commands.

g. Providing an ISR to handle any interrupts that may originate from the board, and to call appropriate DPC's to complete servicing of interrupts.

h. Providing DMA capabilities to move telemetry data from the board's FIFO's to host memory.

i. Unloading the driver and freeing all allocated resources upon demand.

2.2 Context Description

2.2.1 Context Description for the PCI10FR Wrapper API

Figure 2-1 shows a context model for the PCI10FR Wrapper API. The Wrapper API has two software interfaces: one interface to the application, and a second interface to the PCI10FR Device Driver. As mentioned in Section 2.1.1, the interface to the application is fixed. The interface to the PCI10FR Device Driver, however, will change depending on the operating system. This SDD describes the application interface and the Microsoft Windows NT-specific device driver interface.

2.2.2 Context Description for the PCI10FR Device Driver

Figure 2-1 also shows a context model for the PCI10FR Device Driver. The device driver has two interfaces: one software interface to the PCI10FR Wrapper API running in user-mode, and a second interface to the PCI10FR Prototype Board via the PCI bus.

2.3 Context Model

The context model for the PCI10FR Wrapper API and the PCI10FR Device Driver is shown in Figure 2-1.

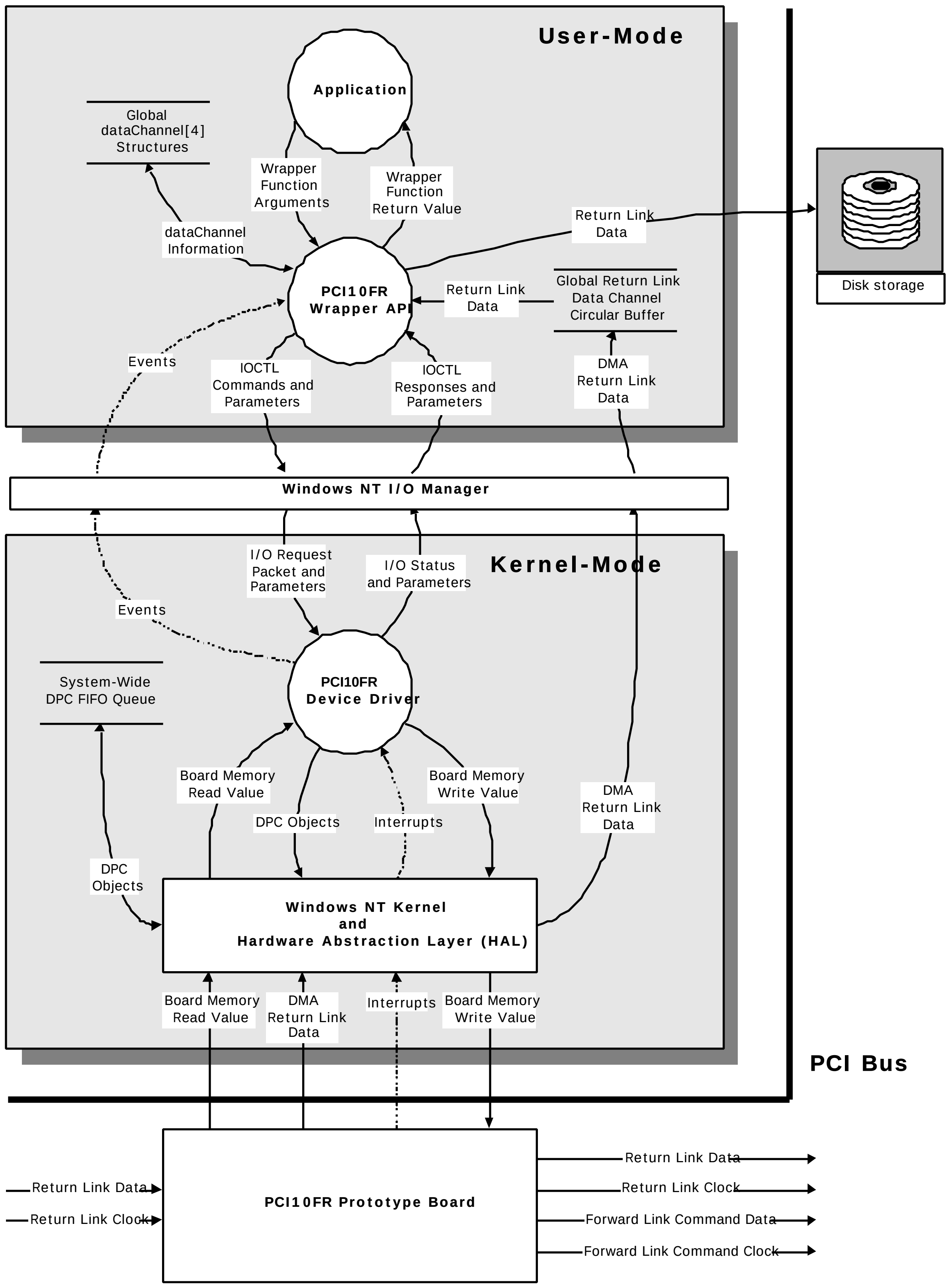


Figure 2-1. PCI10FR Wrapper API and Device Driver Context Model Diagram

2.4 Data Flow Diagrams

Since this document was written after the driver software had already been designed, implemented and tested, the Data Flow Diagrams (DFD's) of the PCI10FR Device Driver are already partitioned into tasks. They are equivalent to the task models described in Section 3. Therefore, rather than present the DFD's in this section and repeat the information in Section 3, all discussions of DFD's and task models are located in Section 3.

2.5 Process Specifications

The process specifications for the DFD's are presented in Section 3 with the DFD's.

2.6 Hardware Control Points

The PCI10FR Device Driver monitors and controls all hardware on the PCI10FR Prototype Board. The device driver provides generic functions of reading and writing to all board memory and PCI configuration space locations. The PCI10FR Wrapper Functions provide a consistent API for user-mode applications to call the device driver's generic functions. The user-mode application is responsible for determining and writing the correct values into the board and chip registers.

A detailed memory map and description of all PCI10FR Prototype Board registers and memory locations is contained in the *PCI10FR Hardware Definition Document*. The hardware control points that the device driver must interface with are listed below:

a. V962PBC: This chip contains 61 PCI configuration space registers. The serial EEPROM on the PCI10FR Prototype Board loads these registers with default values at boot-up, but some of the registers, such as the PCI Interrupt Status, PCI Interrupt Configuration, and the DMA registers must be accessed a lot in the ISR and the DPC routines. All of the registers are completely defined in the *VxxxPBC User's Manual* referenced in Section 1.5.

b. PCI10FR Board Registers: There are five primary board control and status registers on the PCI10FR Prototype Board: the Board Control Register, Status Register 0, Status Register 1, Status Register 2, and the Miscellaneous Control Register. These registers are documented in the *PCI10FR Hardware Definition Document* referenced in Section 1.5.

c. Programmable FIFO Flag Register: This register is used to program the programmable FIFO flags on the four data FIFO's and the one forward link FIFO. This register is documented in the *PCI10FR Hardware Definition Document* referenced in Section 1.5. Information on programming the programmable FIFO flag registers can be found in and email message entitled, *PCI10FR: Programming the FIFO Flags*. A copy of this email message is in Appendix B of this document.

d. PIFS Chip: There are 32 internal PIFS Chip registers. Registers 0 through 17 are read-only status registers; registers 18 through 31 are read/write setup registers. These registers are documented in the *Parallel Integrated Frame Synchronizer Chip* ASIC document referenced in Section 1.5.

e. RSEC Chip: There are 40 internal RSEC Chip registers. Registers 0 through 20 are read-only status registers; registers 21 through 39 are read/write setup registers. These registers are documented in the “Reed-Solomon Error Correction Chip”, in Section 10 of the *Microelectronic Systems Branch Application-Specific Integrated Circuits (ASIC) Components Document* referenced in Section 1.5.

f. RS Routing Table: This 64-Kbyte RAM location on the PCI10FR Prototype Board is used by the RSEC Chip to determine how to route its output frames. This table is documented in the *PCI10FR Hardware Definition Document*, and the “Reed-Solomon Error Correction Chip”, in Section 10 of the *Microelectronic Systems Branch Application-Specific Integrated Circuits (ASIC) Components Document*. Both of these documents are referenced in Section 1.5.

g. Return Link Data FIFO’s and Forward Link Command FIFO: The four return link data FIFO’s and the one forward link command FIFO have addresses in board memory space. The data stored in the FIFO’s can be read directly, and data can be written into these FIFO’s directly. These FIFO’s are documented in the *PCI10FR Hardware Definition Document*, as well as the *CMOS SyncFIFO IDT72241* specification sheets referenced in Section 1.5, and found in Appendix C of this document.

h. NCO Clock Chip: The NCO Clock Chip generates a clock signal for the Forward Link Output Subsystem. The frequency of the clock signal is varied by setup registers in the Board Control Register. This register is documented in *the PCI10FR Hardware Definition Document*. Information on programming the NCO Clock Chip is found in the *Dual Programmable Clock Generator* specification sheets, found in Appendix D of this document, and the *Using CH9203 in Multiple Speed CD-ROM Drive* application notes, found in Appendix E of this document. Both the specification sheets and application notes are published by Chrontel. These documents are referenced in Section 1.5.

i. Temperature Chip: The Temperature Chip is used to monitor board temperature. In actual practice, this chip is not used because the board temperature has not been found to be a problem. The Temperature Chip is programmed and commanded through Miscellaneous Control Register, and the temperature and setup information is read through Status Register 2. These registers are documented in the *PCI10FR Hardware Definition Document*. Information on programming the Temperature Chip can be found in the *DS1620 Digital Thermometer and Thermostat* specification sheets, found in Appendix F, and the *Application Note 105 — High Resolution Temperature Measurement with Dallas Direct-to-Digital Temperature Sensors* application notes, found in Appendix G. Both the specification sheets and application notes are published by Dallas Semiconductor. These documents are referenced in Section 1.5.

j. Serial EEPROM Chip: This chip is used to download PCI configuration space information into the V962PBC on boot-up. The PCI10FR device driver does not really interface directly with this chip, but it could. The Serial EEPROM Chip can be reprogrammed on the board via the V962PBC registers. Information about the Serial EEPROM Chip can be found in the *2-Wire Serial CMOS E^2PROMs* specification, found in Appendix H. This document is published by Atmel, Inc., and it is referenced in Section 1.5.

k. DMA Chaining Memory: There is 128-Kbytes of RAM on the PCI10FR Prototype Board for storing linked addresses for DMA chaining. Unfortunately, Revision B2 of the V962PBC does not support DMA chaining. The V3 Corporation expects to release Revision C0 of the V962PBC sometime in April 1997. This revision will support DMA chaining.

Section 3. Design Description

3.1 Task Model

The task-level design step partitions Data Flow Diagrams (DFD's) into cohesive units of manageable size and complexity called tasks. Since this document was written after the software had already designed, implemented, and tested, the DFD's have already been partitioned into tasks. Therefore, this section of the document presents task-oriented DFD's and their specifications. The context model for the PCI10FR Device Driver and the PCI10FR Wrapper API was presented in Figure 2-1. The Level 1 DFD for the PCI10FR Device Driver is shown in Figure 3-1.

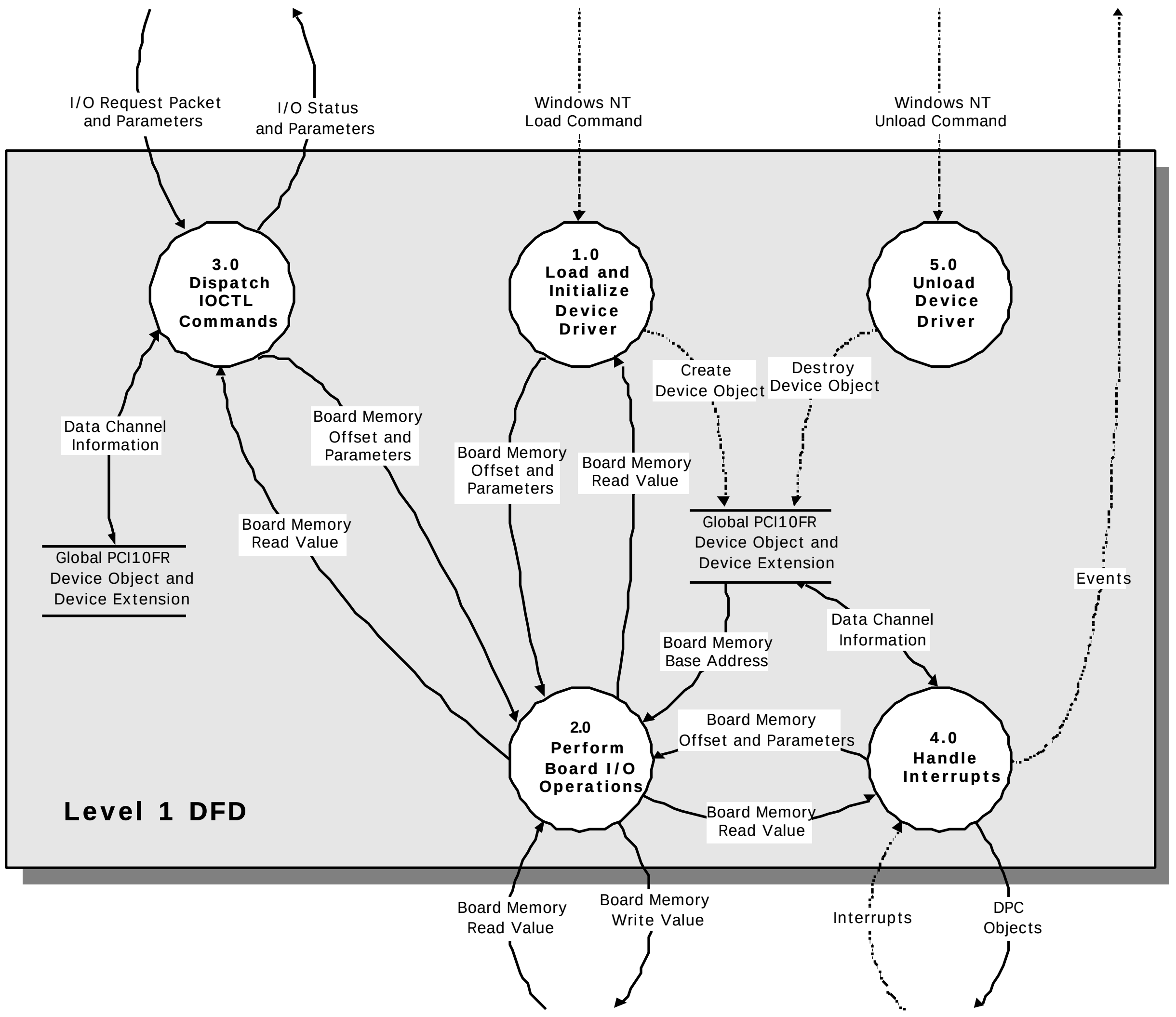

I/O Request Packet and Parameters
I/O Status and Parameters
Windows NT Load Command
Windows NT Unload Command
3.0 Dispatch IOCTL Commands
1.0 Load and Initialize Device Driver
5.0 Unload Device Driver
Create Device Object
Destroy Device Object
Data Channel Information
Board Memory Offset and Parameters
Board Memory Offset and Parameters
Board Memory Read Value
Board Memory Read Value
Global PCI10FR Device Object and Device Extension
Global PCI10FR Device Object and Device Extension
Events
Board Memory Base Address
Data Channel Information
Board Memory Offset and Parameters
2.0 Perform Board I/O Operations
4.0 Handle Interrupts
Board Memory Read Value
Level 1 DFD
Board Memory Read Value
Board Memory Write Value
Interrupts
DPC Objects

Figure 3-1. Level 1 DFD/Task Model for the PCI10FR Device Driver

As shown, the PCI10FR Device Driver has five main tasks:

a. Load and Initialize Device Driver

b. Perform Board I/O Operations

c. Dispatch IOCTL Commands

d. Handle Interrupts

e. Unload Device Driver

Each of these tasks will be presented in detail later in Section 3 of this document.

3.2 Design Assumptions/Dependencies

Several design assumptions were made to simplify implementation of the PCI10FR Device Driver. These assumptions are listed below:

a. The number of data channels and the number of DMA transfer DPC's are limited to four since there are only four return FIFO's on the PCI10FR Prototype Board.

b. The number of DMA Complete DPC's are limited to two since there are only two DMA engines on the V962PBC.

c. The "ProbePci" function was written to look for all instances of PCI10FR boards installed on the PCI bus. Every time this function finds a board, the "CreateDevice" function is called. However, no provisions were made to store information for more than one board.

d. Currently, only three memory resources may be stored in the device extension since the PCI10FR Prototype Board only uses two memory resources. This could be changed by changing the value of "kMaximumMemoryBases" found in the "pci10fr_dev.h" file from 3 to some new value.

e. The vendor identification and device identification for the PCI10FR Prototype Board are currently stored in the "v96xpbc.h" file. Once real PCI vendor and device id's are obtained, they could be stored in the Registry and read by the device driver. This would provide more flexibility.

f. No IOCTL's were created to allow a user-mode application to read from or write to specific PCI configuration registers. It was felt that most applications would not need this capability.

3.3 Device Driver Wrapper API

The PCI10FR Wrapper API provides a consistent interface to applications using the PCI10FR Device Driver, regardless of the platform or operating system being used. Most of the higher level API functions are based on the lower-level read and write functions and are very straight-forward. The three data channel functions are more complicated and will be discussed in their own sections.

The following PCI10FR Wrapper API functions are provided to applications:

a. void OpenDevice(HANDLE *hDriver)

   This function opens the PCI10FR device and provides a handle that is used by all of the other functions to access the device.

b. void CloseDevice(HANDLE hDriver)

   This function closes the PCI10FR device and returns the handle.

c. void ResetBoard(HANDLE hDriver)

   This function resets the PCI10FR Prototype Board.

d. void LoadBoardConfig(HANDLE hDriver, BOOL ld)

   This function loads a binary configuration file containing all of the board, PIFS chip, RSEC chip, and FIFO programmable flag configuration parameters that were previously saved using the "SaveBoardConfig" function. If the second argument is true, the function will prompt the user for the name of the configuration file to load. If it is false, a default configuration file will be loaded.

e. void SaveBoardConfig(HANDLE hDriver, BOOL sv)

   This function saves the board, PIFS chip, RSEC chip, and FIFO programmable flag configuration parameters in a binary configuration file. If the second argument is true, the function will prompt the user for a configuration file name. If it is false, the parameters will be saved in a default configuration file.

f. Uint32 OpenDataChannel(HANDLE hDriver, Uint32 channel, DATA_CHANNEL_PRIORITY priority, Ubyte *filename, Uint32 fileTransferSize, Ubyte *destAddress)

   See Section 3.3.1.

g. Uint32 FlushDataChannels(HANDLE hDriver, BOOLEAN closeChannelFlag)

   See Section 3.3.2.

h. Uint32 CloseDataChannels(HANDLE hDriver)

   See Section 3.3.3.

i. void GetPciConfiguration(HANDLE hDriver)

   This function reads and displays the standard and the V962PBC-specific PCI configuration registers.

j. Uint32 ReadDeviceUbyte(HANDLE hDevice, Uint32 offset, Ubyte buffer)

   This function reads a single 8-bit value into a buffer from a board memory map offset specified by the offset argument. It returns the number of bytes read.

k. Uint32 ReadDeviceUint16(HANDLE hDevice, Uint32 offset, Uint16 buffer)

   This function reads a single 16-bit value into a buffer from a board memory map offset specified by the offset argument. It returns the number of bytes read.

l. Uint32 ReadDeviceUint32(HANDLE hDevice, Uint32 offset, Uint32 buffer)

   This function reads a single 32-bit value into a buffer from a board memory map offset specified by the offset argument. It returns the number of bytes read.

m. Uint32 WriteDeviceUbyte(HANDLE hDevice, Uint32 offset, Ubyte value)

   This function writes a single 8-bit value to a board memory map offset specified by the offset argument. It returns the number of bytes written.

n. Uint32 WriteDeviceUint16(HANDLE hDevice, Uint32 offset, Uint16 value)

   This function writes a single 16-bit value to a board memory map offset specified by the offset argument. It returns the number of bytes written.

o. Uint32 WriteDeviceUint32(HANDLE hDevice, Uint32 offset, Uint32 value)

   This function writes a single 32-bit value to a board memory map offset specified by the offset argument. It returns the number of bytes written.

p. Uint32 ReadDeviceUbyteBuffer(HANDLE hDevice, Uint32 offset, Ubyte buffer, Uint32 numberUchars)

   This function reads a specified number of 8-bit values into a buffer, starting at the board memory map offset specified by the offset argument. It returns the number of bytes read.

q. Uint32 ReadDeviceUint16Buffer(HANDLE hDevice, Uint32 offset, Uint16 buffer, Uint32 numberUshorts)

   This function reads a specified number of 16-bit values into a buffer, starting at the board memory map offset specified by the offset argument. It returns the number of bytes read.

r. Uint32 ReadDeviceUint32Buffer(HANDLE hDevice, Uint32 offset, Uint32 buffer, Uint32 numberUlongs)

   This function reads a specified number of 32-bit values into a buffer, starting at the board memory map offset specified by the offset argument. It returns the number of bytes read.

s. Uint32 WriteDeviceUbyteBuffer(HANDLE hDevice, Uint32 offset, Ubyte buffer, Uint32 numberUchars)

This function writes a specified number of 8-bit values stored in a buffer, starting at the board memory map offset specified by the offset argument. It returns the number of bytes written.

t. Uint32 WriteDeviceUint16Buffer(HANDLE hDevice, Uint32 offset, Uint16 buffer, Uint32 numberUshorts)

   This function writes a specified number of 16-bit values stored in a buffer, starting at the board memory map offset specified by the offset argument. It returns the number of bytes written.

u. Uint32 WriteDeviceUint32Buffer(HANDLE hDevice, Uint32 offset, Uint32 buffer, Uint32 numberUlongs)

   This function writes a specified number of 32-bit values stored in a buffer, starting at the board memory map offset specified by the offset argument. It returns the number of bytes written.

3.3.1 OpenDataChannel Function

This function is used to open a data channel to a specific FIFO on the PCI10FR Prototype Board. Specifically, this function fills in most of the data channel information in the DATA_CHANNEL structure, creates a number of Win32 events, and creates a "ReadDataChannel" thread used to talk directly with the device driver.

The four sections that follow discuss the arguments of the "OpenDataChannel" function, the possible return values of this function, the sequential tasks that this function performs, and the tasks that the "ReadDataChannel" function performs.

3.3.1.1OpenDataChannel Function Arguments

The "OpenDataChannel" function is passed six arguments:

a. A handle to the PCI10FR Device Driver.

b. A channel number from 0 to 3 for the four FIFO banks.

c. A priority argument that is used to set the "ReadDataChannel's" thread priority.

d. A buffer containing a filename to be used for archiving the data to a hard drive. The filename is optional. If no filename is provided, no data archiving will occur.

e. A fileTransferSize argument (in Kbytes) used to determine the size of the file transfers. The fileTransferSize argument is optional only if no filename was provided. If a filename is provided, then this field is necessary.

f. A buffer containing a network destination address in ASCII for networking real-time virtual channels. This field is completely optional since no networking capabilities have been implemented in the software.

3.3.1.2OpenDataChannel Function Return Values

Since there are many sources of errors in creating all of the events, creating the thread, and allocating the circular buffer memory, several error messages can be returned by the "OpenDataChannel" function. Table 3-1 describes these error messages.

Table 3-1. Return Values from the OpenDataChannel Function

Return Value	Description
kNoError	This is the default return value. There were no errors.
kCreateThreadError	An error occurred in creating the "ReadDataChannel" thread.
kOpenEventFileError	Not used anymore. Ignore.
kAllocateBufferError	An error occurred in allocating memory in the "ReadDataChannel" thread.
kCreateEventError	An error occurred in creating at least one event object.
kOpenDataFileError	An error occurred in opening a data archive file in the "ReadDataChannel" thread.
kIoctlOpenError	An error occurred in the "ReadDataChannel" thread when trying to send the device driver an IOCTL open data channel command.

3.3.1.3OpenDataChannel Function Algorithm

The "OpenDataChannel" function performs the following sequential tasks:

a. Zeroes the data channel information stored in the global DATA_CHANNEL structure for the specific data channel.

b. Stores the handle to the driver, the channel number, and the channel priority in the DATA_CHANNEL structure.

c. Determines whether the PIFS chip is in CCSDS or weather mode and stores this information in the DATA_CHANNEL structure. This information is used to determine whether individual virtual channel statistics and counts will be done. These counts are currently performed by one of the debugger functions.

d. If a filename is provided, it is stored in the DATA_CHANNEL structure.

e. If a network destination address is provided, it is stored in the DATA_CHANNEL structure.

f. Calculates the total data frame size based on reading the RSEC output length registers, and based on whether data padding is turned on or off.

g. Initializes all of the data channel memory pointers to NULL.

h. Calculates and stores all of the necessary buffer sizes, DMA transfer sizes, and actual file transfer sizes in the DATA_CHANNEL structure.

i. Creates a kCloseDataChannelEvent and a kFlushDataChannelEvent and stores the event handles in the DATA_CHANNEL structure.

j. Initializes all counters to zero.

k. Creates a new Win32 thread called "ReadDataChannel" at the specified priority level.

l. Checks the "channelEnabled" field in the DATA_CHANNEL structure, which is set by the thread once everything is working, to make sure that the data channel is ready for data transfers.

m. Checks the "errorCode" field in the DATA_CHANNEL structure to see if any errors have occurred in the new thread.

n. Returns a kNoError status if no errors occur. Cleans up everything that was opened, created, or allocated if there is an error message, and then return the error message.

3.3.1.4ReadDataChannel Thread Algorithm

The "ReadDataChannel" thread is created by the "OpenDataChannel" function. Its main purpose is to sleep until awakened by a event, and then to take appropriate action. This thread is passed the specific channel number. It has no return value.

The "ReadDataChannel" thread performs the following event-driven tasks:

a. Allocates a large buffer based on the buffer size information found in the DATA_CHANNEL structure. Returns

b. Divides the buffer into DMA blocks and event blocks. A DMA block is the amount of data actually transferred from the FIFO into the buffer per DMA transfer. For efficiency, the device driver does not signal a data event for every DMA transfer. It waits until enough DMA blocks have been transferred to make an efficient file transfer. It then sets a data event and the entire event block, made up of numerous DMA blocks, is archived to disk.

c. Creates a kBufferOverflowEvent and numerous data events. The kBufferOverflowEvent is used by the device driver to signal that the circular buffer has overflowed. The data events are used by the device driver to signal that an event block is ready to be archived to disk.

d. If a filename was provided, the thread opens a data file to archive the data.

e. If a network destination address was provided, the thread does nothing. Again, this feature is not implemented.

f. Sends an open data channel IOCTL command to the device driver.

g. If anything has gone wrong up to this point, the thread reverses all the previous activities and returns with an appropriate error message. If everything worked, the thread goes to sleep in an event loop waiting for one of the events to occur.

h. If a data event occurs, the data is archived to the open archive file, if applicable.

i. If the data is CCSDS data, a PCI10FR Debugger function called “gatherStats” is called to gather individual virtual channel statistics and counts.

j. Resets the data event.

k. Updates the event block counters and pointers.

l. If a kFlushDataChannelEvent occurs, the event is reset and a flush data channel IOCTL command is sent to the device driver. This IOCTL call returns the number of 32-bit words that were flushed into the circular buffer. This data is then archived (if applicable), CCSDS statistics are gathered (if applicable), and the counters and pointers are updated.

m. If a kCloseDataChannelEvent occurs, the event is reset and a close data channel IOCTL command is sent to the device driver. The archive file is closed (if applicable), all of the events are closed, the circular buffer memory is freed, the “channelEnabled" field in the DATA_CHANNEL structure is set to FALSE, and the thread exits the event loop and returns.

n. If a kBufferOverflowEvent occurs, the event is reset and the numberBufferOverflows counter in the DATA_CHANNEL structure is incremented. Currently, the thread takes no further action other than record the number of overflows.

3.3.2 FlushDataChannels Function

This function is used to flush and capture all of the return link data that is left in all of the data channel FIFO’s after a mission. The function can also be called during a pass if there is a need to “start over”.

The “FlushDataChannels” function is passed a handle to the PCI10FR Device Driver, as well as a BOOLEAN flag indicating whether the function was called from the “CloseDataChannels” function (true), or whether the function was called directly (false). The “FlushDataChannels” function performs the following tasks:

a. Checks to make sure that at least one data channel is open, and returns immediately if there are no open data channels. In this case, the return value is kNoDataChannelsOpenError.

b. Disables the PIFS chip so that new data cannot enter the system.

c. Sleeps for 100 milliseconds to allow data to empty out of the PIFS and RSEC chips, due to time-outs, and into the data channel FIFO’s.

d. For each open data channel, sets the kFlushDataChannelEvent. This will cause the “ReadDataChannel” thread to awaken and perform the necessary action. The “FlushDataChannels” function checks the flushComplete field in the DATA_CHANNEL structure to make sure that each data channel has been flushed before setting the next data channel’s kFlushDataChannelEvent.

e. If the “FlushDataChannels” function was not called by the “CloseDataChannels” function, then the PIFS chip is reenabled. Otherwise, the PIFS chip is left disabled.

f. Returns kNoError if there were no errors.

3.3.3 CloseDataChannels Function

This function is used to flush and close all of the open data channels. The “CloseDataChannels” function is passed a handle to the PCI10FR Device Driver, and it then performs the following tasks:

a. Calls the “FlushDataChannels” function with the closeChannelFlag parameter set to true. If an error occurs in the “FlushDataChannels” function, the error messages described in Section 3.3.2 are forwarded as the “CloseDataChannels” function returns immediately.

b. For each open data channel, sets the kCloseDataChannelEvent. This will cause the “ReadDataChannel” thread to awaken and perform the necessary action. The “CloseDataChannels” function checks the channelEnabled field in the DATA_CHANNEL structure to make sure that each data channel has been closed before setting the next data channel’s kCloseDataChannelEvent.

c. As each data channel is closed, its “ReadDataChannel” thread returns, the thread id is closed, and the specific data channel information in the DATA_CHANNEL structure is zeroed.

d. Returns kNoError if there were no errors.

3.4 Device Driver Data Structures

There are several very important data structures used in the PCI10FR Device Driver. These data structures include:

a. The PCI_DEVICE structure is defined in the “pci10fr_dev.h” file. This is the global device extension structure of the PCI10FR device object. It is used to store all static variables, such as the data channel information, the board’s base memory address, the interrupt vectors, etc.

b. The V3_SPECIFIC_PCI_CONFIG structure is defined in the “v96xpbc.h” file. It contains the V962PBC-specific PCI configuration register values.

c. The DATA_CHANNEL structure is defined in the “pci10frnt.h” file. This structure is used by both the PCI10FR Device Driver and the PCI10FR Wrapper API to store data channel information.

d. The RW_PARAMETERS structure is defined in the “pci10frnt.h” file. This structure is used with read and write IOCTL commands to pass board memory map offsets and write values to the device driver.

e. The PCI_COMMON_CONFIG structure is defined in the Windows NT “ntddk.h” header file. This structure contains the standard PCI configuration register values. It is used in several driver functions to store the current values of the PCI configuration registers. It is also used by the PCI10FR Wrapper API in the “GetPciConfiguration” function.

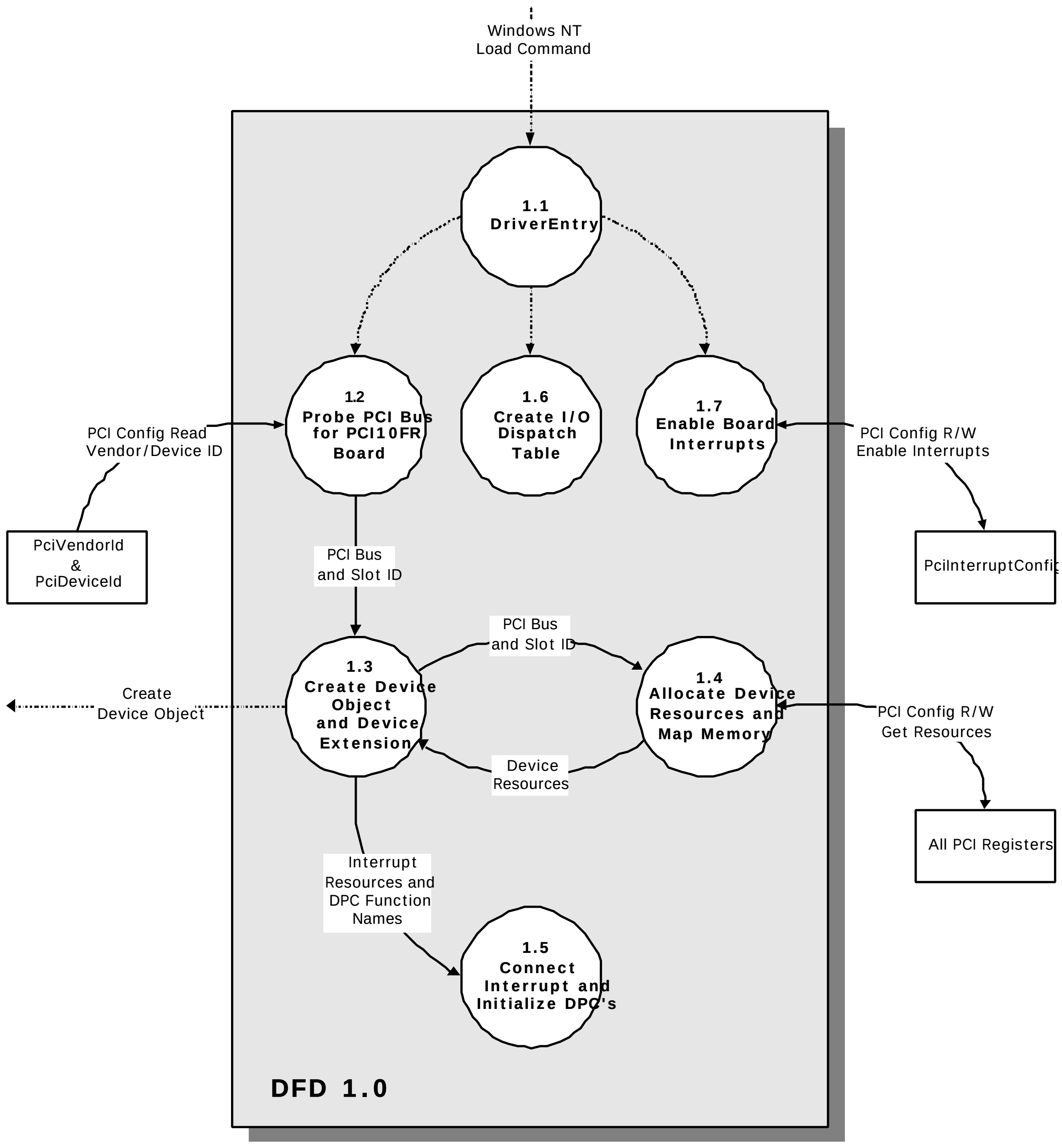

Windows NT Load Command
1.1 DriverEntry
1.2 Probe PCI Bus for PCI10FR Board
1.6 Create I/O Dispatch Table
1.7 Enable Board Interrupts
PCI Config Read Vendor/Device ID
PciVendorId & PciDeviceId
PCI Config R/W Enable Interrupts
PciInterruptConfig
PCI Bus and Slot ID
PCI Bus and Slot ID
1.3 Create Device Object and Device Extension
1.4 Allocate Device Resources and Map Memory
Create Device Object
PCI Config R/W Get Resources
Device Resources
All PCI Registers
Interrupt Resources and DPC Function Names
1.5 Connect Interrupt and Initialize DPC's
DFD 1.0

Figure 3-2. Load and Initialize Device Driver Task

3.5 Device Driver Loading and Initialization Task

Before a device driver can be used by an application, it must first be registered with the Windows NT Registry. A device driver only needs to be registered once, unless the name of the driver or the start up parameters are changed. Registration is discussed in Section 4.4.

After a device driver is registered, it must be loaded into memory and initialized. Information about when a driver is loaded is contained in the "ini" file used during registration. A device driver may be loaded manually, or it may be loaded automatically. The "ini" file tells Windows NT exactly when to load the driver. In the case of the PCI10FR Device Driver, the "pci10fr.ini" file tells the operating system to load the driver manually. Therefore before the driver is used, the following command must be entered at the command prompt:

```
net start pci10fr
```

This command causes the Windows NT I/O Manager to call the DriverEntry function of the PCI10FR Device Driver, as shown in the DFD in Figure 3-2.

The DriverEntry function of the device driver must first determine whether a PCI10FR device is installed on the PCI bus. It does this by calling a function called ProbePci. This function examines every board on the PCI bus, or busses, by reading their PCI configuration registers and comparing their vendor identifications and device identifications with the identifications for the PCI10FR board. Once the ProbePci function finds a board that matches the PCI10FR board, it calls a function called, CreateDevice.

NOTE

> Currently, the PCI10FR is still using the vendor identification and device identification of the V962PBC. Since this board is a prototype, no vendor or device identification numbers were requested from the PCI Special Interest Group (SIG). Before the PCI10FR Board can be sold as a commercial product, new identification numbers must be obtained.

Most of the work of initializing the PCI10FR Device Driver occurs in the CreateDevice function. This function performs the following sequential tasks:

a. Creates a device object called "pci10fr". It also creates a device extension, which is nothing more than device-specific global memory for the device driver.

b. Creates a symbolic link, or name, that a Windows NT application can specify to open the device and get a handle to it. In this case, the name is "pci10fr".

c. Calls the ResetBoard function, which in turn calls the FlushV962PbcFifos function and the ReinitBoard function. The FlushV962PbcFifos function flushes the board's V962PBC read and write FIFO's to eliminate any garbage data lingering in them.

d. The ReinitBoard function reinitializes some of the PCI configuration registers to their correct values. This step is no longer necessary when using Revision B2 of the V962PBC. Earlier revisions required that the V962PBC registers be unlocked at boot up, and unfortunately allowed some read-only registers to be overwritten.

e. Uses the Hardware Abstraction Layer (HAL) call, "HalAssignSlotResources", to obtain a list of all required resources. These resources include memory, port I/O, and interrupts. The PCI10FR Board requires two resources: a 16-Mbyte memory address space, and an interrupt vector. A third resource, a port I/O resource, is also obtained for reading and writing to the PCI configuration space; however, this resource is not needed since there are direct HAL functions that handle PCI configuration I/O.

f. Allocates the memory resource required, and translates the bus address into a physical address, and then into a system address that can be used by the device driver. This memory address is stored in the driver's device extension and is used in reading and writing to board memory locations.

g. Allocates an interrupt vector for the board.

h. Initializes a couple of Windows NT Adapter objects for use in DMA operations.

i. Calls the EnableBoard function to enable the board's memory space and bus master control bits in the V962PBC Control Register.

j. Initializes four deferred procedure calls (DPC's), one per return link FIFO, for use in transferring data from these FIFO's via DMA.

k. Initializes two DPC's, one per DMA engine, for use in completing a DMA transfer.

l. Connects the interrupt vector.

If any one of these steps fails, the PCI10FR Device Driver will reverse everything that has been done up to the point of error, and then return STATUS_UNSUCCESSFUL to the Windows NT operating system.

After the CreateDevice function has performed its many tasks, the DriverEntry function will create dispatch points for all legitimate calls made to the device driver. The Windows NT I/O Manager uses these dispatch points to call the correct functions within the device driver.

Finally, the board interrupts are enabled by writing to the V962PBC's PCI Interrupt Configuration register. The two DMA interrupts and the local board interrupt are the only two interrupts that are enabled. The PCI10FR does not use the mail-box interrupts. In retrospect, it would be a good idea not to enable board interrupts until all data channels are open and the system is ready to go.

All of these functions discussed above are located in the "pci10fr.c" file.

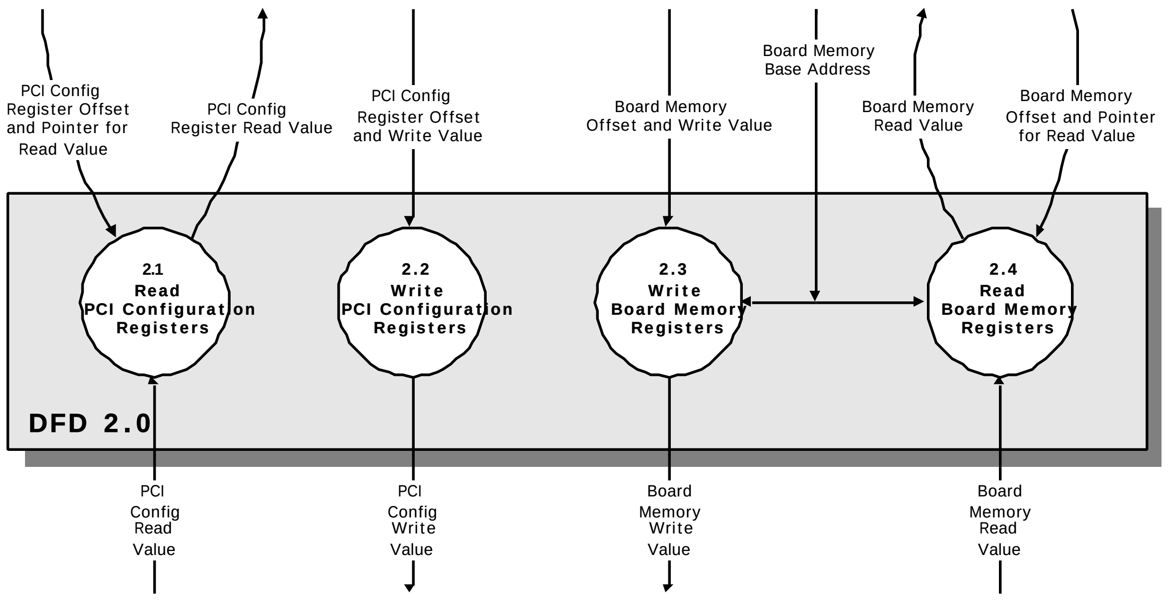

PCI Config Register Offset and Pointer for Read Value
PCI Config Register Read Value
PCI Config Register Offset and Write Value
Board Memory Offset and Write Value
Board Memory Base Address
Board Memory Read Value
Board Memory Offset and Pointer for Read Value
2.1 Read PCI Configuration Registers
2.2 Write PCI Configuration Registers
2.3 Write Board Memory Registers
2.4 Read Board Memory Registers
DFD 2.0
PCI Config Read Value
PCI Config Write Value
Board Memory Write Value
Board Memory Read Value

Figure 3-3. Board I/O Task

3.6 Device Driver Board I/O Task

The Board I/O task, which is shown in Figure 3-3, is used by all other tasks to read from and write to PCI configuration registers, and to read from and write to all PCI10FR Board registers and memory spaces. The Board I/O task uses Windows NT HAL calls to perform its task.

3.6.1 Reading PCI Configuration Registers

Three functions are provided to read PCI configuration space, depending on whether the value to be read is an 8-bit, a 16-bit, or a 32-bit value:

a. ReadPciConfigUchar (8-bit value)

b. ReadPciConfigUshort (16-bit value)

c. ReadPciConfigUlong (32-bit value)

The arguments passed to these functions include the PCI10FR device extension, the offset within PCI configuration space that is to be read, and a pointer to a buffer to hold the value read. These functions then use the operating system’s “HalGetBusDataByOffset” function to read the value into the buffer. Each of the three functions returns the number of bytes actually read.

3.6.2 Writing PCI Configuration Registers

Three functions are also provided to write to PCI configuration space:

a. WritePciConfigUchar (8-bit value)

b. WritePciConfigUshort (16-bit value)

c. WritePciConfigUlong (32-bit value)

The arguments passed to these functions include the PCI10FR device extension, the offset within PCI configuration space that is to be written to, and a pointer to a buffer that holds the value to be written. These functions then use the operating system's "HalSetBusDataByOffset" function to write the value from the buffer into the PCI configuration register. Each of the three functions returns the number of bytes actually written.

Two additional functions are provided to unlock and lock the lockable PCI configuration registers: UnlockPciRegisters and LockPciRegisters.

All of the functions used to read from and write to PCI configuration registers, as well as the UnlockPciRegisters and LockPciRegisters functions, are located in the "rwconfig.c" file.

3.6.3 Reading Board Memory Locations

Reading and writing to board memory locations is easy and is done so frequently that specific functions were not developed for this task. When another driver function needs to read a board memory location, it adds the local bus memory map offset of the register to the MemoryBase address stored in the PCI10FR device extension. It then performs one of six possible memory read operations provided by the HAL:

a. READ_REGISTER_UCHAR (reads a single 8-bit value)

b. READ_REGISTER_USHORT (reads a single 16-bit value)

c. READ_REGISTER_ULONG (reads a single 32-bit value)

d. READ_REGISTER_BUFFER_UCHAR (reads a specified number of 8-bit values)

e. READ_REGISTER_BUFFER_USHORT (reads a specified number of 16-bit values)

f. READ_REGISTER_BUFFER_ULONG (reads a specified number of 32-bit values)

The first three functions are used when only a single value needs to be read from the board. The argument to these three functions is the board address that was calculated from the register offset and MemoryBase address. The functions return the actual value.

The second three functions are used when multiple contiguous board memory locations need to be read. The arguments to these three functions are the calculated board address, a pointer to a buffer to store the values read, and the number of values to be read. These functions do not have a return value. Note that the number of values to be read is not the same as the number of bytes. For example, if one wanted to read 15 ULONG's from the board, the correct number of values to read argument for the READ_REGISTER_BUFFER_ULONG function is 15 values — not 60 bytes.

3.6.4 Writing Board Memory Locations

Writing to board memory locations is very similar to reading from board memory locations. When another driver function needs to write to a board memory location, it adds the local bus memory map offset of the register to the MemoryBase address stored in the PCI10FR device extension. It then performs one of six possible memory write operations provided by the HAL:

a. WRITE_REGISTER_UCHAR (writes a single 8-bit value)

b. WRITE_REGISTER_USHORT (writes a single 16-bit value)

c. WRITE_REGISTER_ULONG (writes a single 32-bit value)

d. WRITE_REGISTER_BUFFER_UCHAR (writes a specified number of 8-bit values)

e. WRITE_REGISTER_BUFFER_USHORT (writes a specified number of 16-bit values)

f. WRITE_REGISTER_BUFFER_ULONG (writes a specified number of 32-bit values)

The first three functions are used when only a single value needs to be written to the board. The arguments to these three functions are the board address that was calculated from the register offset and MemoryBase address, and the value to be written. The functions do not return any value.

The second three functions are used when multiple contiguous board memory locations need to be written. The arguments to these three functions are the calculated board address, a pointer to a buffer that stores the values to be written, and the number of values to be written. These functions also do not have a return value. Note that the number of values to be written is not the same as the number of bytes. For example, if one wanted to write 15 ULONG's to the board, the correct number of values to write argument for the WRITE_REGISTER_BUFFER_ULONG function is 15 values — not 60 bytes.

All of the board read and write functions are available to an application program through IOCTL calls to the PCI10FR Device Driver. These IOCTL calls are processed by the Dispatch Task.

3.7 Device Driver Dispatch IOCTL Commands Task

The Dispatch IOCTL Commands Task is responsible for handling all driver calls from a user-mode application. During the Loading and Initialization Task, the DriverEntry function of the device driver stores dispatch points (function pointers) in the driver object, which is later used by the Windows NT I/O Manager to determine which function to call in response to a user-mode application I/O request. When a user-mode application makes a Win32 device driver call, the Windows NT I/O Manager creates an I/O Request Packet (IRP), looks at its dispatch table, and calls the appropriate function within the device driver. The device driver is then responsible for handling the IRP and quickly returning status through the IRP back to the user-mode application. This is shown in the Dispatch IOCTL Commands DFD in Figure 3-4.

A user-mode application must first open a device and get a handle to the device object before it can make any other calls to the device. The Win32 API function for opening a device is called CreateFile. It is the same function that is used to open a file. The PCI10FR device is opened with a device name of "pci10fr", with generic read and write attributes, and with asynchronous operations turned off. The Win32 API is well documented and is not included in this document. The Win32 CreateFile function is encapsulated by the PCI10FR Wrapper API call, "OpenDevice".

Once a handle to the PCI10FR device is obtained, the user-mode application may issue IOCTL commands to the PCI10FR Device Driver. All of the PCI10FR Device Driver IOCTL definitions are included in the "pci10frnt.h" file. This file is common to both the PCI10FR Device Driver and user-mode applications such as the PCI10FR Debugger. The PCI10FR Wrapper API encapsulates the underlying "DeviceIoControl" Win32 API call.

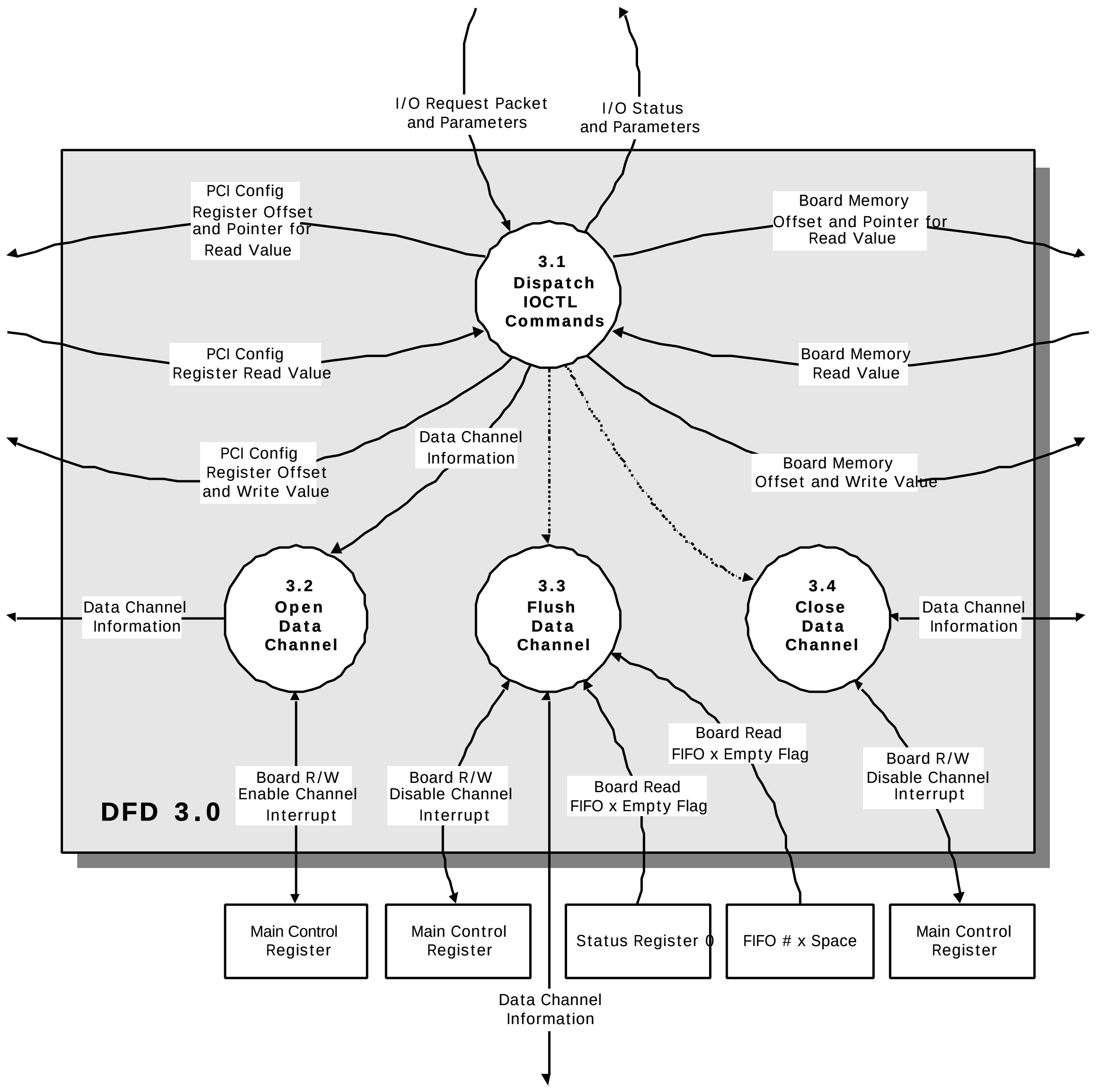

I/O Request Packet and Parameters
I/O Status and Parameters
PCI Config Register Offset and Pointer for Read Value
Board Memory Offset and Pointer for Read Value
3.1 Dispatch IOCTL Commands
PCI Config Register Read Value
Board Memory Read Value
PCI Config Register Offset and Write Value
Data Channel Information
Board Memory Offset and Write Value
Data Channel Information
3.2 Open Data Channel
3.3 Flush Data Channel
3.4 Close Data Channel
Data Channel Information
Board Read FIFO x Empty Flag
Board R/W Enable Channel Interrupt
Board R/W Disable Channel Interrupt
Board Read FIFO x Empty Flag
Board R/W Disable Channel Interrupt
DFD 3.0
Main Control Register
Main Control Register
Status Register 0
FIFO # x Space
Main Control Register
Data Channel Information

Figure 3-4. Dispatch IOCTL Commands Task

The Dispatch IOCTL Commands Task is handled by two functions that are included in the "dispatch.c" file: "Dispatch" and "IoctlDispatch". The first function handles the two basic non-IOCTL calls, "IRP_MJ_CREATE" and "IRP_MJ_CLOSE". The second function handles all of the PCI10FR Device Driver-specific IOCTL calls, including:

a. IOCTL Read Commands

b. IOCTL Write Commands

c. IOCTL Open Data Channel Command

d. IOCTL Flush Data Channel Command

e. IOCTL Close Data Channel Command

Each of these commands is discussed in the following sections.

3.7.1 IOCTL Read Commands

As seen in Figure 3-4, the Dispatch IOCTL Commands task (DFD bubble 3.1) allows a user-mode application to request board memory reads and writes of 8-bit, 16-bit, and 32-bit values. The IoctlDispatch function uses the Board I/O Task to actually perform the board read and write operations. This capability is used extensively to setup the board registers and to get board status.

For board memory read operations, the user-mode application sends the device driver one of seven IOCTL read commands:

a. IOCTL_PCI10FR_READ_UCHAR (reads a single 8-bit value)

b. IOCTL_PCI10FR_READ_USHORT (reads a single 16-bit value)

c. IOCTL_PCI10FR_READ_ULONG (reads a single 32-bit value)

d. IOCTL_PCI10FR_READBUFFER_UCHAR (reads a specified number of 8-bit values)

e. IOCTL_PCI10FR_READBUFFER_USHORT(reads a specified number of 16-bit values)

f. IOCTL_PCI10FR_READBUFFER_ULONG (reads a specified number of 32-bit values)

g. IOCTL_PCI10FR_GET_CONFIGURATION_SPACE (reads all of the entire PCI configuration registers on the V962PBC)

Included in the IOCTL read command is a pointer to a variable to hold the read value or values, the number of values to read, and a parameter structure (RW_PARAMETERS) that contains the board memory map offset for the location to read. Once the user-mode application issues the IOCTL command, the I/O Manager generates an IRP, looks up and calls the appropriate dispatch function within the PCI10FR Device Driver, and passes the IRP to it. The IoctlDispatch function's switch construct determines which IOCTL command was issued, extracts the parameters, uses the Board I/O Task's functions to carry out the command, stores the read value or values in the read variable's location, stores the status of the request in the IRP, and completes the I/O request by calling the Windows NT "IoCompleteRequest" function.

The last IOCTL read command is used to read the PCI configuration registers on the V962PBC. This function is used primarily for debugging purposes. Additional PCI configuration read and write IOCTL commands can easily be added to the device driver.

3.7.2 IOCTL Write Commands

For board memory write operations, the user-mode application sends the device driver one of six IOCTL write commands:

a. IOCTL_PCI10FR_WRITE_UCHAR (writes a single 8-bit value)

b. IOCTL_PCI10FR_WRITE_USHORT (writes a single 16-bit value)

c. IOCTL_PCI10FR_WRITE_ULONG (writes a single 32-bit value)

d. IOCTL_PCI10FR_WRITEBUFFER_UCHAR (writes a specified number of 8-bit values)

e. IOCTL_PCI10FR_WRITEBUFFER_USHORT(writes a specified number of 16-bit values)

f. IOCTL_PCI10FR_WRITEBUFFER_ULONG (writes a specified number of 32-bit values)

Included in the first three IOCTL write commands are the number of values to write, and a parameter structure (RW_PARAMETERS) that contains the board memory map offset for the location to write, and the actual value to write. Included in the second three IOCTL write commands are a pointer to a variable that holds the values to be written, the number of values to write, and a parameter structure (RW_PARAMETERS) that contains the board memory map offset for the location to write. As with the read operations, once the user-mode application issues the IOCTL command, the I/O Manager generates an IRP, looks up and calls the appropriate dispatch function within the PCI10FR Device Driver, and passes the IRP to it. The IoctlDispatch function's switch construct determines which IOCTL command was issued, extracts the parameters, uses the Board I/O Task's functions to carry out the command, stores the status of the request in the IRP, and completes the I/O request by calling the Windows NT "IoCompleteRequest" function.

3.7.3 IOCTL Open Data Channel Command

The IOCTL_PCI10FR_OPEN_DATA_CHANNEL command is used by the PCI10FR Wrapper API "ReadDataChannel" thread to prepare the PCI10FR Device Driver to start receiving return link telemetry data from one of the four return link data FIFO's on the board. This IOCTL command passes the device driver a DATA_CHANNEL structure containing all of the pertinent information about the data channel, such as the channel (FIFO) number, the base virtual address for the circular buffer, data transfer lengths, and a set of event handles to be used by the device driver to signal the "ReadDataChannel" thread. This IOCTL command causes the device driver to sequentially:

a. Extract the data channel information from the IRP and store this information in its device extension, as shown in Figure 3-4.

b. Translate the base virtual address pointer into a Windows NT Memory Description List (MDL) using the "MmCreateMdl" function.

c. Probe and lock the user-mode pages of memory so that this memory cannot be paged out during a DPC. The "MmProbeAndLockPages" function is used to perform this task.

d. Translate the MDL into a virtual memory address for DMA purposes, and into a system address for flushing purposes. The "MmGetMdlVirtualAddress" and "MmGetSystemAddressForMdl" functions are used to perform this task.

e. Initialize the current DMA block pointer to the base virtual address and the current data event to the first possible data event.

f. Translate the events handles passed in the DATA_CHANNEL into pointers for use by the device driver.

g. Enable the specific channel’s FIFO interrupt by writing to the board’s Main Control Register.

h. Completing the IRP.

In retrospect, it would make more sense not to enable the specific channel’s interrupt in this IOCTL command, but rather have separate IOCTL commands to enable and disable channel interrupts. The problem with the current method is that data may start flowing and causing interrupts before all of the data channels have been “opened”.

3.7.4 IOCTL Flush Data Channel Command

This IOCTL command is issued by the “ReadDataChannel” thread in response to a “FlushDataChannels” PCI10FR Wrapper API call. The “FlushDataChannels” call disables the PIFS chip so that new data cannot be received, and it sequentially issues flush events to every open data channel. Both the PIFS chip and the RSEC chip have time-out features so that any data left in these chips will automatically be flushed into the designated FIFO. Each channel’s “ReadDataChannel” thread awakens when this event occurs and issues an IOCTL_PCI10FR_FLUSH_DATA_CHANNEL command for its particular data channel. The thread passes the device driver a current copy of the DATA_STRUCTURE.

With the PIFS chip now turned off, and all of the data in the PIFS chip and RSEC chip now flushed into the designated FIFO, this IOCTL command causes the device driver to sequentially:

a. Extract the data channel information from the, as shown in Figure 3-4.

b. Disable the channel’s FIFO interrupt by writing to the board’s Main Control Register.

c. Read the channel’s FIFO one 32-bit word at a time and storing this data in the user-mode circular buffer until Status Register 0 shows that the FIFO is empty.

d. Updating the pointers and the number of 32-bit words that have been flushed.

e. Reenabling the channel’s interrupt.

f. Completing the IRP and returning the number of 32-bit words flushed.

If the “FlushDataChannels” call was issued directly, the PIFS chip is reenabled. If the “FlushDataChannels” call was issued by the “CloseDataChannels” call, the PIFS chip is left disabled.

3.7.5 IOCTL Close Data Channel Command

This IOCTL command is issued by the “ReadDataChannel” thread in response to a “CloseDataChannels” PCI10FR Wrapper API call. The “CloseDataChannels” call actually calls the “FlushDataChannels” function first with a parameter telling the “FlushDataChannels” function not to reenable the PIFS chip. The “CloseDataChannels” function then sequentially issues close events to every open data channel. Each channel’s “ReadDataChannel” thread awakens when this event occurs and issues an IOCTL_PCI10FR_CLOSE_DATA_CHANNEL command for its particular data channel. The thread passes the device driver a current copy of the DATA_CHANNEL structure.

With the PIFS chip now turned off, and all of the data in the PIFS chip, the RSEC chip, and the FIFO now flushed, this IOCTL command causes the device driver to sequentially:

a. Extract the data channel information from the, as shown in Figure 3-4.

b. Disable the channel’s FIFO interrupt by writing to the board’s Main Control Register.

c. Unlocking the locked memory pages for the specific channel using the “MmUnlockPages” function.

d. Freeing the MDL using the “IoFreeMdl” function.

e. Reinitializing all of the DATA_CHANNEL pointers to NULL.

f. De-referencing all of the event objects for the specific channel.

g. Zeroing out all of the DATA_CHANNEL memory for the specific channel.

h. Completing the IRP.

Once the IRP has been completed, the “ReadDataChannel” thread will perform clean up operations to close files and free memory before returning and thus closing the thread.

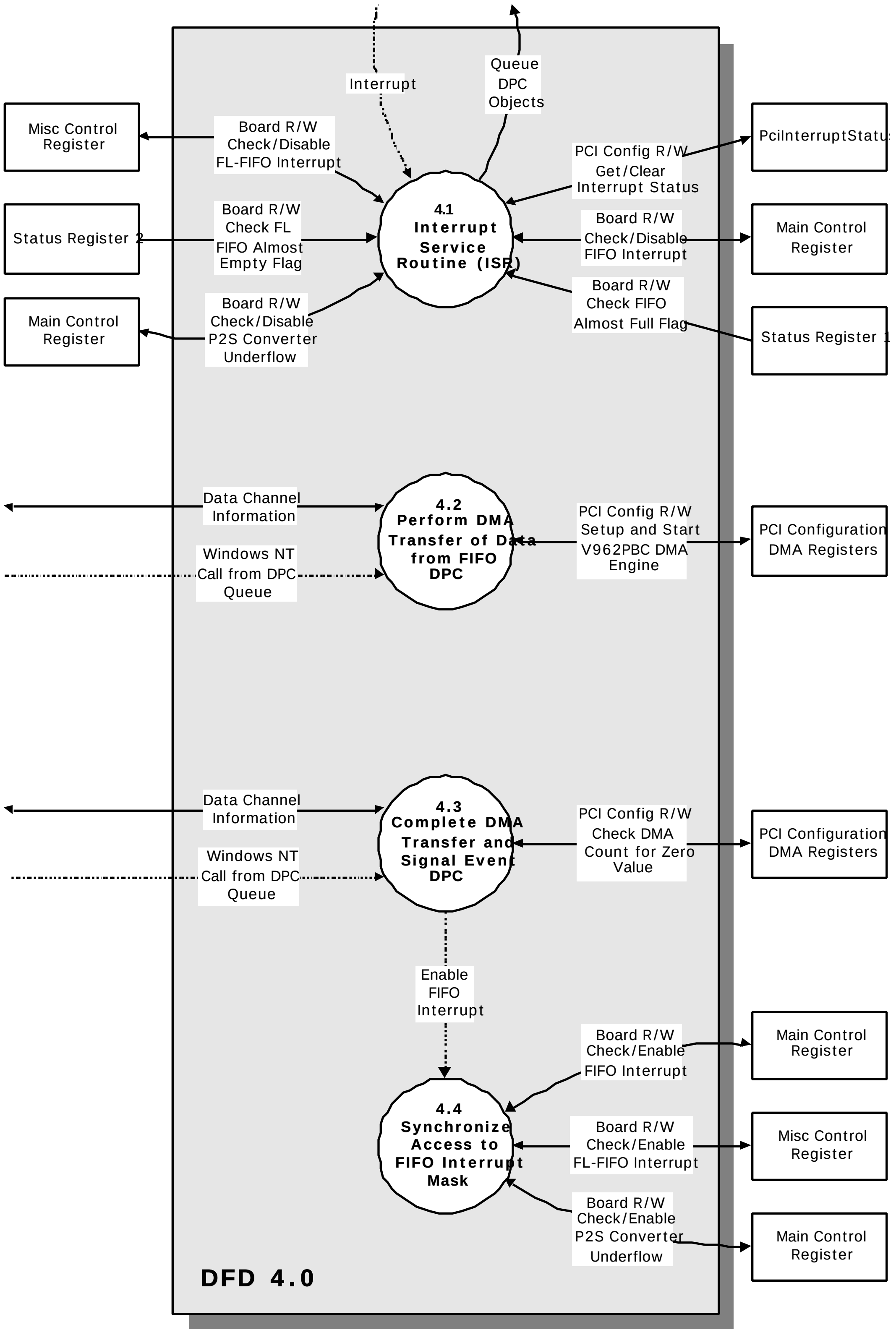

Interrupt
Queue DPC Objects
Misc Control Register
Board R/W Check/Disable FL-FIFO Interrupt
PCI Config R/W Get/Clear Interrupt Status
PciInterruptStatu
Status Register 2
Board R/W Check FL FIFO Almost Empty Flag
4.1 Interrupt Service Routine (ISR)
Board R/W Check/Disable FIFO Interrupt
Main Control Register
Main Control Register
Board R/W Check/Disable P2S Converter Underflow
Board R/W Check FIFO Almost Full Flag
Status Register 1
Data Channel Information
4.2 Perform DMA Transfer of Data from FIFO DPC
PCI Config R/W Setup and Start V962PBC DMA Engine
PCI Configuration DMA Registers
Windows NT Call from DPC Queue
Data Channel Information
4.3 Complete DMA Transfer and Signal Event DPC
PCI Config R/W Check DMA Count for Zero Value
PCI Configuration DMA Registers
Windows NT Call from DPC Queue
Enable FIFO Interrupt
Board R/W Check/Enable FIFO Interrupt
Main Control Register
4.4 Synchronize Access to FIFO Interrupt Mask
Board R/W Check/Enable FL-FIFO Interrupt
Misc Control Register
Board R/W Check/Enable P2S Converter Underflow
Main Control Register
DFD 4.0

Figure 3-5. Interrupt Services Task

3.8 Device Driver Interrupt Services Task

The Interrupt Services Task (IST) is responsible for handling all interrupts originating from the PCI10FR Prototype Board. Since Windows NT allows shared interrupts, the IST must also quickly forward any interrupts that do not belong to the PCI10FR Prototype Board. The IST conforms to the standard Windows NT device driver protocol of performing very little in the interrupt service routine (ISR) and having the ISR queue deferred procedure calls (DPC's) as needed to complete the work at a much lower interrupt priority level.

Although the Forward Link FIFO and the P2S converter can generate interrupts, these interrupts are generally masked off. The ISR does check for these interrupts, but currently it does nothing more than disable them. There are currently no DPC's for handling these two interrupt conditions. The rest of this section describes the typical return link FIFO servicing operation. This operation can be broken down into five phases:

a. Idle Phase

b. FIFO Service ISR Phase

c. DMA Transfer DPC Phase

d. DMA Engine Service ISR Phase

e. DMA Complete DPC Phase

3.8.1 Idle Phase

After a data channel has been opened, and while no DMA is in progress, the device driver software waits for enough data to accumulate in one or more of the four 32-bit FIFO banks to generate an interrupt.

3.8.2 FIFO Service ISR Phase

Upon receiving an interrupt, the interrupt service routine (ISR) first checks the V962PBC's PCI Interrupt Status Register, as shown in Figure 3-5, to make sure that the interrupt originated from the board. If the interrupt is not from the board, the ISR returns quickly and forwards the interrupt back to the Windows NT operating system.

If the interrupt did originate on the board, the ISR determines whether the interrupt is due to completion of a DMA operation or due to other causes on the board. If the interrupt was not due to completion of a DMA operation, then Status Registers 1 and 2 are read to determine which of the on-board sources of interrupt occurred. If one of the four FIFO banks generated the interrupt due to the almost-full flag being asserted, the ISR disables the FIFO's interrupt, queues a FIFO-specific DPC to initiate a DMA operation to move data from the FIFO into host memory, clears the V962PBC chip's PCI Interrupt Status Register, and returns from the interrupt. Since any number of the FIFO's may need servicing at a given time, the ISR checks all of them whether they generated the interrupt or not. This makes the ISR much more efficient.

3.8.3 DMA Transfer DPC Phase

Since the Windows NT operating system will not allow the same DPC object to be in the system-wide queue (FIFO) at the same time, four DMA transfer DPC objects are used — one for each data channel (FIFO). When the ISR is complete, Windows NT lowers the interrupt priority level down to Dispatch Level. It then checks to see if there are any DPC's on the queue. If there are, then the first DPC on the queue gets called. If there are no DPC's in the queue, then the interrupt priority level is returned to Normal Level.

Assuming that one of the DMA transfer DPC's are called by the operating system, the DPC performs the following sequential tasks, as shown in Figure 3-5:

a. Checks to see if which, if any, DMA engine is currently freed up for servicing the FIFO. If both DMA engines are in use, the DPC queues itself and returns.

b. If a DMA engine is available, the DPC calculates the number of map registers needed for the data transfer.

c. Once the number of map registers has been determined, the DPC calls the Windows NT "IoAllocateAdapterChannel" function to allocate an adapter channel, and to call the "AdapterControl0" or "AdapterControl1" functions (depending on which DMA engine is being used) to actually do the work of setting up the DMA engine.

d. The "AdapterControl0" or "AdapterControl1" function flushes the Windows NT I/O buffers, obtains a physical address for the circular buffer by calling the "IoMapTransfer" function, sets up the V962PBC DMA engine for a DMA transfer, and then initiates the transfer.

e. Once the "IoAllocateAdapterChannel" function returns, then this DPC returns.

3.8.4 DMA Engine Service ISR Phase

When the DMA operation has completed, an interrupt will be generated and the ISR will once again determine the cause. If the interrupt is due to a DMA operation completing, the ISR will clear the DMA interrupt through the V962PBC chip's PCI Interrupt Status Register and the Local Bus Interrupt Control and Status Register. A DPC will then be queued to complete the transfer.

3.8.5 DMA Complete DPC Phase

The DMA complete DPC performs the following functions:

a. Checks the DMA count register to make sure that it is zero.

b. Flushes the adapter buffers using the "IoFlushAdapterBuffers" function.

c. Frees the map registers.

d. Updates the memory pointers and counters in the DATA_CHANNEL structure.

e. Asynchronously notifies the user application that data has been successfully transferred into host memory. It does this by setting the data event objects. If the specific data event object is already set, then the "ReadDataChannel" thread has not serviced and cleared the memory space. In the case, a buffer overflow event is signaled to inform the "ReadDataChannel" thread. Note that this overflow event does not signal that the FIFO overflowed, but rather that the circular buffer overflowed. Currently, there is no way to detect a FIFO overflow condition without polling the unlatched flag register.

f. Free the DMA engine so that it may be used again.

g. Reenable the FIFO's interrupt. This must be synchronized with the ISR. Four synchronization functions — one for each FIFO — are used to synchronize enabling of each FIFO's interrupt.

h. Return.

Once the DPC returns, the device driver goes back into an Idle phase.

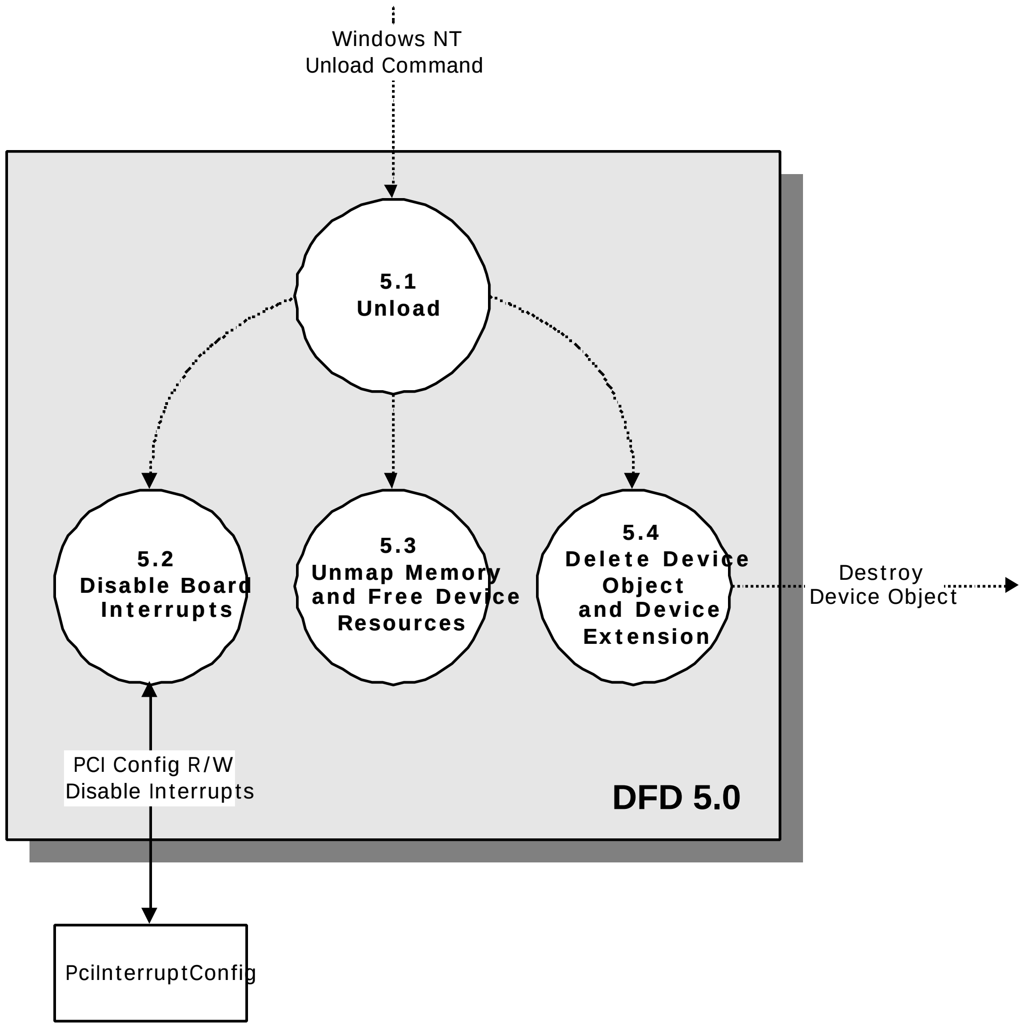

Windows NT Unload Command
5.1 Unload
5.2 Disable Board Interrupts
5.3 Unmap Memory and Free Device Resources
5.4 Delete Device Object and Device Extension
Destroy Device Object
PCI Config R/W Disable Interrupts
DFD 5.0
PciInterruptConfig

Figure 3-6. Unload Device Driver Task

3.9 Device Driver Unloading Task

Unloading a device driver frees up all resources and actually removes the device driver from memory. To unload the PCI10FR Device Driver, type the following command at a command prompt:

```
net stop pci10fr
```

This command will cause the Windows NT I/O Manager to call the device driver's "Unload" function. The "Unload" function, which is shown in Figure 3-6, performs the following tasks:

a. Deletes the symbolic links to the device driver.

b. Disables the PCI10FR Prototype Board's interrupts.

c. Disconnects the interrupt vector using the "IoDisconnectInterrupt" function.

d. Dereferences all event objects stored in the device extension.

e. Unlocks all mapped memory and frees all MDL's.

f. Unmaps the board memory from system space.

g. Unreports the resources that were assigned.

h. Deletes the device object.

3.10 Device Driver Error Handling

The PCI10FR Device Driver checks for errors during the Load and Initialization Task. If any errors occur, a print statement for the Microsoft WinDbg program is generated, all of the preceding functions are reversed, all resources are freed, and the driver returns a STATUS_UNSUCCESSFUL message to the operating system.

In the Dispatch IOCTL Commands Task, if an unknown IOCTL command is requested, the device driver will immediately return a STATUS_INVALID_PARAMETER error message. When the open data channel IOCTL command is requested, an exception trap is used to trap any exceptions raised when trying to probe and lock memory pages. If an exception does occur, the MDL will be freed and a STATUS_ACCESS_VIOLATION error message will be returned.

Section 4. Implementation Data

4.1 Coding and Naming Conventions

All coding conventions and practices conform to the *Software and Automation Systems Branch C++ Style Guide*, referenced in Section 1.5, to the maximum extent possible considering that the PCI10FR Wrapper API, the PCI10FR Device Driver, and the PCI10FR Debugger were all written in C.

4.2 File Descriptions

The source code and header files for the PCI10FR project can be broken down into three main file categories: files necessary to build the PCI10FR Wrapper API by itself; files necessary to build the PCI10FR Device Driver; and files necessary to build the PCI10FR Debugger application, which includes the PCI10FR Wrapper API. These files are described in the sections below.

4.2.1 File Descriptions for the PCI10FR Wrapper API

These are the files necessary to compile the PCI10FR Wrapper API:

a. PCIFRNT.H (header file) contains the IOCTL definitions for the making calls to the PCI10FR Device Driver, the DATA_CHANNEL structure definition, and all of the function prototypes for the PCI10FR Wrapper API

b. DRIVERTYPES.H (header file) contains a set of define statements that create standard names for variable types so that the PCI10FR Wrapper API can be used on other platforms and operating systems, such as the Dec Alpha workstation.

c. OFFSETS.H (header file) contains a set of define statements for all of the PCI10FR Prototype Board memory map offsets.

d. ROUTINES.H (header file) contains all of the function prototypes for the PCI10FR Wrapper API, as well as the PCI10FR Debugger application.

e. WRAPPER.H (header file) contains all of the PCI10FR Wrapper API function prototypes contained in WRAPPER.C.

f. PCI.H (header file) contains the Microsoft Windows NT PCI configuration space structure definition.

g. CHANNEL.H (header file) contains define statements and function prototypes for the CHANNEL.C file.

h. WRAPPER.C (source file) contains all of the PCI10FR Wrapper API functions, except for the data channel functions and the load/save configuration functions..

i. CHANNEL.C (source file) contains all of the data channel functions.

j. CONFIG.C (source file) contains the load and save configuration functions.

4.2.2 File Descriptions for the PCI10FR Device Driver

These are the files necessary to build the PCI10FR Device Driver:

a. NTDDK.H (header file) contains the data structures and prototypes provided by Microsoft in the DDK to build Windows NT device drivers.

b. STDARG.H (header file) defines ANSI-style macros for variable argument functions. This header file is also provided by Microsoft.

c. PCIFRNT.H (header file) is the same file as mentioned in Section 4.2.1.

d. PCI10FR_DEV.H (header file) contains the device extension structure for the PCI10FR device object that is created in CreateDevice, as well as all of the function prototypes for the PCI10FR Device Driver.

e. V96XPBC.H (header file) contains define statements for all of the PCI configuration register offsets in the V962PBC, as well as a structure definition for the V962PBC PCI configuration space.

f. INTERRUPT.H (header file) contains all of the define statements used in the INTERRUPT.C file

g. PCI10FR.C (source file) contains all of the initialization functions that are called during the loading of the PCI10FR Device Driver.

h. DISPATCH.C (source file) contains the dispatch switch statements that handle IOCTL calls from the user-mode application.

i. RWCONFIG.C (source file) contains all of the functions that read and write to PCI configuration space.

j. INTERRUPT.C (source file) contains the ISR, its associated DPC's, and synchronization functions.

k. SOURCES (type of makefile) contains information necessary to build the PCI10FR Device Driver.

l. MAKEFILE (makefile) contains the makefile used to build the PCI10FR Device Driver. Both the SOURCES and MAKEFILE must be in the same directory as the header and source files in order for the build utility to work properly.

4.2.3 File Descriptions for the PCI10FR Debugger Application

These are the files necessary to build the PCI10FR Debugger application:

a. DRIVERTYPES.H (header file) is the same file as mentioned in Section 4.2.1.

b. PCI.H (header file) is the same file as mentioned in Section 4.2.1.

c. PCIFRNT.H (header file) is the same file as mentioned in Section 4.2.1.

d. OFFSETS.H (header file) is the same file as mentioned in Section 4.2.1.

e. ROUTINES.H (header file) is the same file as mentioned in Section 4.2.1.

f. NCO.H (header file) contains all of the define statements and structures necessary to configure the Chrontel NCO Clock Chip.

g. FIFOFLAG.H (header file) contains all of the define statements and structures necessary to read and to program the FIFO programmable flags.

h. RSROUTETBL.H (header file) contains the structure definition for the RS Routing Table entries.

i. CHANNEL.H (header file) is the same file as mentioned in Section 4.2.1

j. WRAPPER.H (header file) is the same file as mentioned in Section 4.2.1.

k. STATSTR.H (header file) contains define statements and structure definitions for the Virtual Channel statistics counts performed in the STATS.C file.

l. RSMACROS.H (header file) contains macro definitions for bit-wise manipulation of RSEC Chip registers.

m. PIFSMACROS.H (header file) contains macro definitions for bit-wise manipulation of PIFS Chip registers.

n. BOARDMACROS.H (header file) contains macro definitions for bit-wise manipulation of PCI10FR board registers.

o. DEBUG.C (source file) contains the main function, and provides the main menu for performing all PCI10FR Debugger functions.

p. WRAPPER.C (source file) is the same file as mentioned in Section 4.2.1.

q. SETUP.C (source file) contains a variety of board setup functions.

r. CONFIG.C (source file) is the same file as mentioned in Section 4.2.1.

s. PIFSCHIP.C (source file) contains all of the functions that talk with the PIFS Chip.

t. RSCHIP.C (source file) contains all of the functions that talk with the RSEC Chip.

u. FIFO.C (source file) contains functions to directly read the return link data FIFO’s.

v. FIFOFLAG.C (source file) contains functions to program the FIFO programmable flag registers.

w. LOOPBACK.C (source file) contains functions to write data directly into the PIFS Chip and have it flow through the system.

x. CHANNEL.C (source file) is the same file as mentioned in Section 4.2.1.

y. NCO.C (source file) contains functions to program the NCO Clock Chip.

z. SCANBUF.C (source file) contains functions to scan an archived telemetry data file and check for sync words and correct Virtual Channel sequence counts.

aa. STATS.C (source file) contains functions to gather, count, and display Virtual Channel statistics.

bb. DEBUG.MAK (makefile) is the makefile for the PCI10FR Debugger application. This file is generated by the Microsoft Visual C++ Developer Studio application.

cc. DEBUG.MDP (project file) is the project or workspace file for the PCI10FR Debugger application. This file is generated by the Microsoft Visual C++ Developer Studio application.

dd. DEBUG.NCB (network control block) is created and used by the Microsoft Visual C++ Developer Studio application when a new project or workspace file is generated.

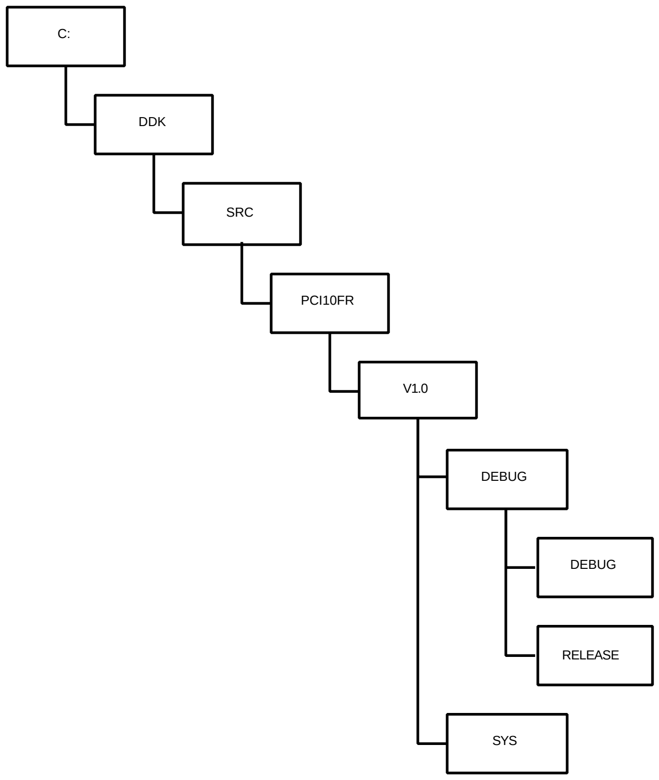


Figure 4-1. Build Directory Structure

4.3 Building the Device Driver

Building the PCI10FR Device Driver under Windows NT 3.51 requires several Microsoft products, including:

a. Microsoft Visual C++ 4.0

b. Microsoft Software Development Kit (SDK) - available through a subscription to the Microsoft Developer Network.

c. Microsoft DDK - available through a subscription to the Microsoft Developer Network.

Although there are other ways to build a Windows NT device driver, such as using C++ libraries offered by third-party vendors and non-Microsoft compilers, this document will only describe the "official Microsoft" method.

All of the PCI10FR Device Driver files should be placed in a subdirectory under the "\ddk\src" directory. The directory structure used to develop the PCI10FR Device Driver and the PCI10FR Debugger is shown in Figure 4-1. A "pci10fr" directory was created off of the "\ddk\src" directory. Each version of the software was given its own directory. The PCI10FR Device Driver files were stored in the "pci10fr\v1.0\sys" subdirectory, and the PCI10FR Debugger files were stored in the "pci10fr\v1.0\debug" subdirectory. All debug versions of the Debugger were stored in the "debug\debug" subdirectory, and the release versions the Debugger were stored in the "debug\release" subdirectory.

There are two options when building the PCI10FR Device Driver. One can build a checked version of the driver, which contains debug information, or one can build a free version of the driver. The choice is made by double-clicking the appropriate icon in the Windows NT DDK program group.

Once the MS-DOS Console window appears, change to the directory containing all of the PCI10FR Device Driver files listed above in Section 4.2.2. At the DOS prompt, type:

```
build -cef
```

The build function uses the nmake function of Microsoft Visual C++ to compile and link the driver into a "sys" file. Any errors are displayed on the screen. A "build.log" file is also generated that contains all of the details of the build, including all warning and error messages.

After a successful build, the "pci10fr.sys" file must be moved or copied into the "drivers" directory. For example, if one created a free version of the driver, one would type:

```
copy c:\ddk\lib\i386\free\pci10fr.sys c:\winnt351\systems32\drivers
```

For a checked version of the driver, one would type:

```
copy c:\ddk\lib\i386\checked\pci10fr.sys c:\winnt351\systems32\drivers
```

Once the driver has been moved into the "drivers" directory, it must be registered. Registration of a device driver only needs to be done once as long as subsequent versions of the device driver do not change the name of the driver or the names used in the CreateDevice function. The next section describes the registration process.

Once the PCI10FR Device Driver has been registered and the computer has been rebooted, the driver may be started. Starting the device driver may be done in several ways. One can type the following command at an MS-DOS Console window:

```
net start pci10fr
```

Similarly, the device driver may be stopped by typing:

```
net stop pci10fr
```

Alternatively, one can open the “Drivers” utility under the “Control Panel” and manually start the PCI10FR Device Driver from there. This utility can also be used to change when the driver is loaded. It can be set to load automatically on boot-up.

WARNING

Always remember to unload a loaded PCI10FR Device Driver before copying a new version to the “drivers” directory. Failure to do so will cause the system to crash!

Further information about building and debugging a device driver can be found in the Win32 DDK documentation referenced in Section 1.5.

4.4 Registering the Device Driver

After the PCI10FR Device Driver has been built and copied in the “drivers” directory, it must be registered in the Windows NT Registry. This registration process is only required the first time a new driver is introduced, as long as the name of the driver and the names used in the CreateDevice function do not change.

There is a file called “PCI10FR.INI” that is located in the same directory as the source files for the PCI10FR Device Driver. This “ini” file contains information about when and how the PCI10FR Device Driver should be loaded. Currently, this file tells the operating system that the driver will be loaded manually. Rather than change this file, it is much easier to register the device driver using the current file, and make any changes to the method of loading in the “Drivers” utility under the Control Panel applet.

To register the PCI10FR Device Driver, change directories to the directory containing the pci10fr.ini file and type:

```
regini pci10fr.ini
```

The “regini” utility is provided in the DDK and is located in the “\ddk\bin” directory. If the DDK has been properly installed, this directory is included in the PATH configuration. Once the “regini” utility has been run, shutdown the computer and restart it. The PCI10FR Device Driver should now be ready to be loaded.

Further information about registering a device driver can be found in the Win32 DDK documentation referenced in Section 1.5.

4.5 Building the Device Driver Wrapper API and Debugger

A Microsoft Developer Project (mdp) file has been created for the PCI10FR Debugger. Assuming that all of the PCI10FR Debugger files have been placed in the directory structure shown in Figure 4-1, one can open the PCI10FR Debugger workspace within the Microsoft Visual C++ Developer Studio application. The files within the PCI10FR Debugger workspace may be edited, rebuilt and debugged using the standard procedures provided by the Developer Studio application. The PCI10FR Wrapper API files may also be compiled only and not linked into the PCI10FR Debugger application.

After the PCI10FR Debugger application has been built, one can start it from an MS-DOS Console window or from the Developer Studio application. If one starts it from an MS-DOS Console window, one needs to first change to the directory containing the debug.exe file, and then type:

debug

Appendix A contains a brief overview of the PCI10FR Debugger application.

NOTE

The PCI10FR Device Driver must already be loaded before starting the PCI10FR Debugger application. If it is not, the Debugger will immediately exit with an error message.

4.6 Error Code Definitions

Windows NT uses a standard set of status/error codes for its device driver function calls. Table 4-1 lists some of the more common definitions. A complete list of Windows NT status/error returns can be found in the "ntstatus.h" file found in the Win32 DDK include directory.

Table 4-1. Standard Windows NT Status/Error Codes

Status/Error Codes	Definition
STATUS_SUCCESS	Value returned to the PCI10FR Wrapper API if the IOCTL call was successful.
STATUS_UNSUCCESSFUL	Value returned to the PCI10FR Wrapper API if the IOCTL call was unsuccessful.
STATUS_INVALID_PARAMETER	Value returned to the PCI10FR Wrapper API if the parameters passed in the IOCTL call were invalid.
STATUS_ACCESS_VIOLATION	Value returned to the PCI10FR Wrapper API if the device driver was unsuccessful in probing and locking down user-mode memory pages to be used in DMA data transfers.

Section 5. Development Issues

5.1 Hardware Issues

5.1.1 V962PBC PCI Bridge Chip Revisions

There have been three revisions of the V962PBC since the project began: Revision B0, Revision B1, and Revision B2. A copy of each V962PBC Change Notice has been provided in Appendix I. Revision B2 solves almost all of the problems encountered in the previous revisions. However, none of the B revisions support DMA chaining. Revision C0, which should be available in April 1997, will be the first V962PBC to support this feature.

5.1.2 Short PCI Clock Signal Trace

The PCI clock signal trace that goes directly from the PCI bus connector to the V962PBC is too short. The PCI specification (Version 2.1) states that this trace must be 2.5-inches ± 0.1-inches from the connector pad to a component. One of the PCI10FR Prototype Boards has had the clock signal trace cut and a 2-inch wire inserted.

5.1.3 FIFO Overflow

There is no way of knowing whether a return link data FIFO has overflowed without polling the full-flag bits in the Board Control Register. These bits are not latched, so even checking them in the ISR when servicing other data FIFO's is a hit-or-miss proposition.

5.1.4 Local Bus CPU-Emulation PLD's

Several PLD's are used to emulate chip select, acknowledge, and other CPU signals on the i960 local bus on the PCI10FR Prototype Board. Unfortunately, these PLD's add two extra clock cycles in performing a read command, thus degrading system performance especially during DMA transfers. In addition, the PLD controlling the read/write lines to the RSEC have problems with bursts of write data to the RSEC. For this reason, the RSEC chip registers must be loaded one at a time with separate write commands. The buffer write command (WriteDeviceUint32Buffer) cannot be used since it will cause the RSEC chip registers to be corrupted.

5.1.5 Gateway 2000 Ultra-SCSI Disk Problem

The PCI10FR Prototype Board works fine up to 15 Mbps on a Gateway 2000 150 MHz Pentium-Pro computer with a Fast SCSI hard drive, and on a Dell OptiPlex GXpro 200 MHz Dual Pentium-Pro computer with an Ultra SCSI hard drive. However, a problem has been noted on a Gateway 2000 200 MHz Pentium-Pro computer with an Ultra SCSI hard drive. Telemetry data from the board is successfully transferred to host memory via DMA, but somehow the data gets corrupted when writing it to the hard drive. Garbage bytes are introduced in the data file. Both the Dell and Gateway 2000 200 MHz computers are using Adaptec 2940 Ultra SCSI adapters with the same setup parameters and the same driver. This problem has not been fully explored due to time constraints. It may just be a faulty disk or motherboard.

5.2 Software Issues

5.2.1 Porting the PCI10FR Device Driver to Windows NT 4.0

The PCI10FR Device Driver and the PCI10FR Wrapper API have been developed and tested under the Windows NT 3.51 operating system. The software should port directly to Windows NT 4.0 without any changes other than changing the SDK and DDK to the Windows NT 4.0 versions. However, the software has not been ported yet because the Windows NT 4.0 operating system, Service Packs, and driver development environment are still relatively new and bug-ridden.

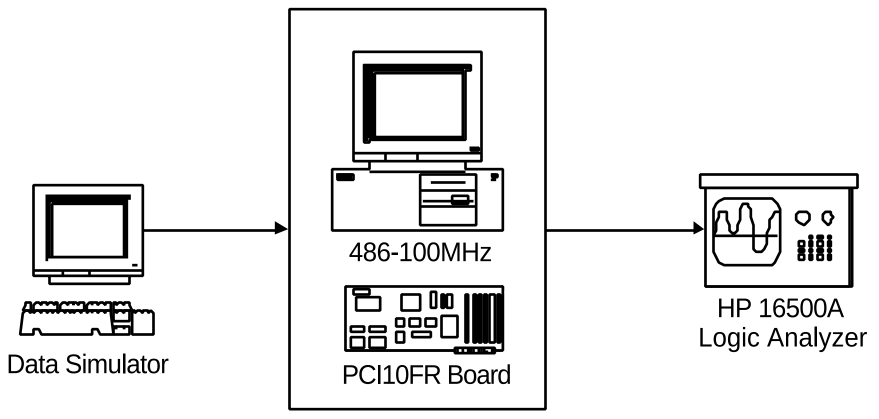
5.2.2 Processing Weather Data Formats in the PIFS Chip

When processing weather data formats, it is very important to write a non-zero value in the 4-bit Word Size B (Wdszb) field (bits 28 - 31) of the PIFS Chip Setup Register 20 even if a second sub-block is not required. Failure to do so will result in either one or no frames of weather data being output from the PIFS Chip.

5.3 Windows NT Performance Issues

One of the reasons for developing the PCI10FR Prototype Board was to measure the performance of Windows NT as a soft real-time operating system (RTOS). Most of the systems developed within Code 521 have used a true RTOS, like VxWorks. After conducting performance tests on the PCI10FR Prototype Board and measuring ISR and DPC latencies, the conclusion is that Windows NT 3.51 is barely adequate for this purpose.

Figure 5-1 shows one of the performance test setups that was used to measure ISR and DPC latencies using programmed I/O to move data from one of the return link data FIFO's to host memory and to archive this data on the hard drive. Figure 5-2 shows the timing diagram that resulted from this test. The four programmable bits in the Miscellaneous Control Register are linked to test points on the board. One of these bits was set in the ISR as soon as it was entered, and was subsequently cleared on exiting the routine. Another bit was set in the DPC on entry, and was cleared on exiting the routine. Table 5-1 shows the results of this performance test using programmed I/O data transfers.



10.0 Mbps
232 bytes / frame

Version 10.0 pci10fr.sys
test232.cnf
Uses FIFO1 only
FIFO1 AF Flag = 1024
256 32-bit words per DPC

F1FF	= FIFO1 Full Flag
F1AF	= FIFO1 Almost Full Flag
ISR	= Interrupt Service Routine
DPC	= Deferred Procedure Call
F1OE	= FIFO1 Output Enable
F1WEN	= FIFO1 Write Enable

Figure 5-1. Performance Test Setup for Programmed I/O Data Transfers

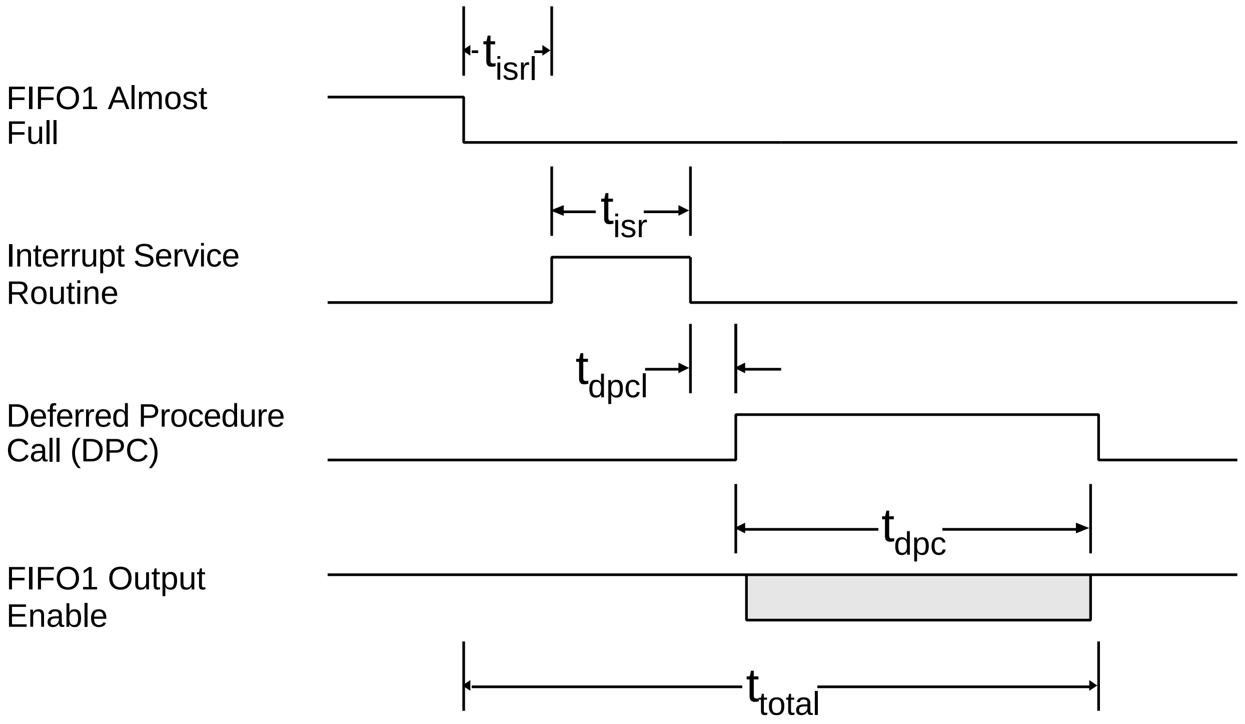


Figure 5-2. Data Transfer Timing Diagram Using Programmed I/O

Table 5-1. Performance Test Results Using Programmed I/O

Time Element	Minimum (μsecs)	Maximum (μsecs)	Average (μsecs)
t-isrl	9.280	81.80	10.89
t-isr	44.32	110.2	47.81
t-dpcl	7.680	39.68	8.470
t-dpc	145.4	177.1	152.8
t-total	206.68	408.78	219.97

Figure 5-3 shows another performance test setup that was used to measure ISR and DPC latencies using DMA transfers of data from one of the return link data FIFO's to host memory. Again, this data was also archived on the hard drive. Figure 5-4 shows the resulting timing diagram. Again, one of the programmable bits was set on entry in the ISR, and cleared on exiting the routine. Another bit was set in the DmaReadFifo1Dpc routine, which initiates a DMA transfer. The bit was not cleared until a second DPC, the Dma0ReadCompleteDpc, had completed. This DPC is called by the ISR when the DMA operation is complete. Table 5-2 shows the results of this performance test using DMA data transfers. The numbers in the Minimum column reflect a one-time measurement and not a true minimum. No maximum or average times were obtained for some of the parameters. However, maximum and average times were obtained for the length of the DPC. These values are shown in their respective columns.

Note that only 256 32-bit words were transferred per DPC using programmed I/O, versus 2900 32-bit words transferred using DMA. The effective average transfer rate using programmed I/O is 0.86 μsecs/32-bit word, while the effective transfer rate using DMA is 0.97 μsec/32-bit word.

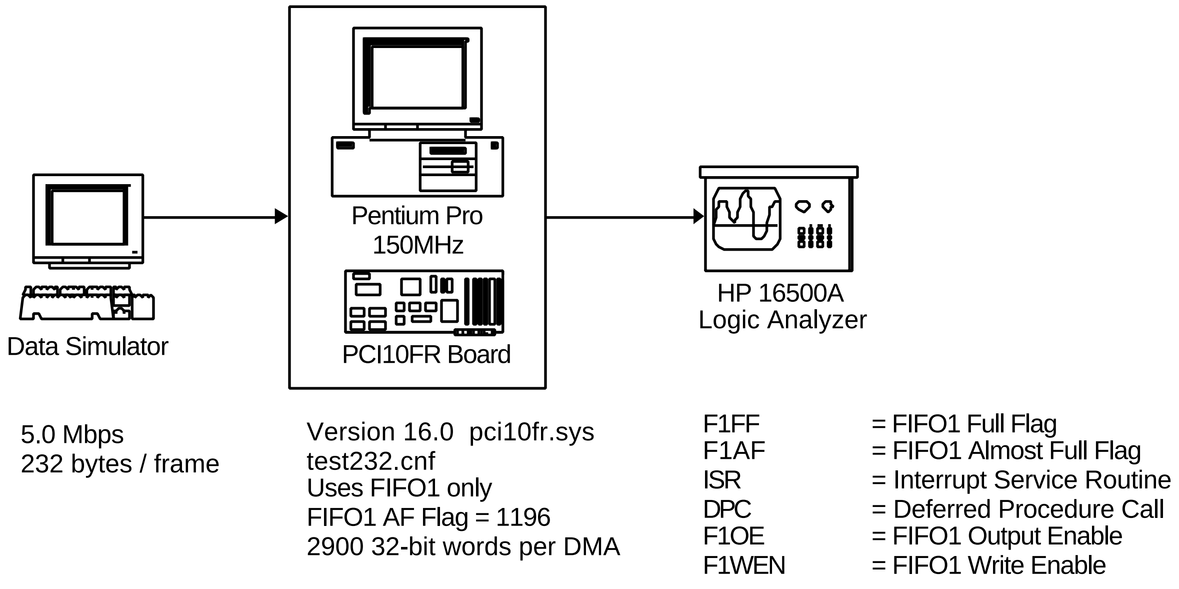


Figure 5-3. Performance Test Setup for DMA Data Transfers

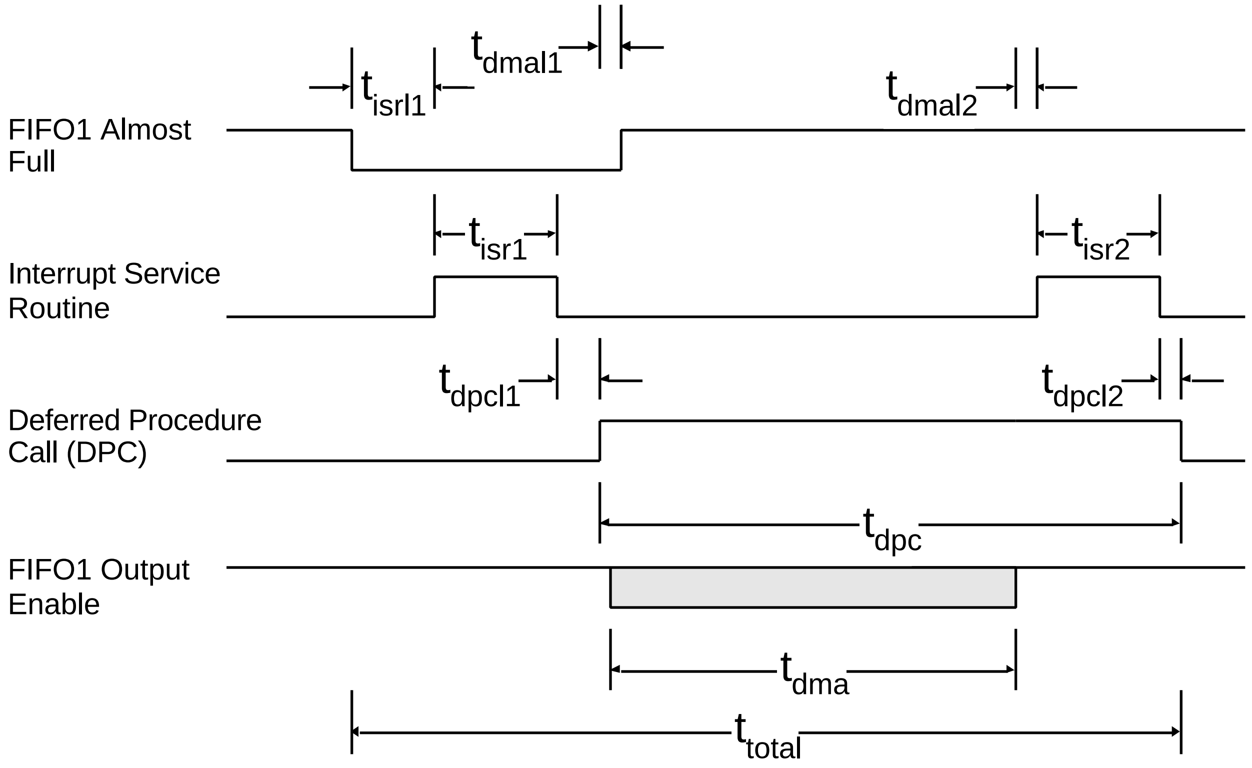


Figure 5-4. Data Transfer Timing Diagram Using DMA

Table 5-2. Performance Test Results Using DMA

Time Element	Minimum (μsecs)	Maximum (μsecs)	Average (μsecs)
t-isrl1	6.201		
t-isr1	35.00		
t-dpcl1	7.000		
t-dmal1	97.00		
t-dma	2295		
t-dmal2	39.84		
t-isr2	30.24		
t-dpcl2	295.8		
t-dpc	2757.9	4534	2882
t-total	2806.1		

These test results indicate that moving telemetry data using DMA transfers takes about the same amount of time as using programmed I/O. There are several reasons for this.

First, as was discussed in Section 5.1.4, the CPU-emulation PLD's on the PCI10FR Prototype Board add two extra wait cycles for every read. This seriously degrades the performance of the board when transferring data via DMA.

Second, since Revision B2 of the V962PBC does not support DMA chaining, the PCI10FR Device Driver uses the standard Windows NT map registers for emulating DMA chaining. Windows NT maps the non-contiguous user-mode memory space into a dedicated area of contiguous system memory. During a DMA cycle, telemetry data is transferred via DMA from a return link data FIFO into this contiguous memory, effectively avoiding the need for DMA chaining. Windows NT then copies this data from the contiguous memory into the user-mode memory space. The advantage of this method is that Windows NT can provide support for boards that do not support DMA chaining without multiple interrupts. The disadvantage is that this method does not support true DMA. The CPU is still involved in the process of copying the data from one memory location to another. Improved performance should be achieved once Revision C0 of the V962PBC, which directly support DMA chaining, is installed on the board.

One can see by looking at Table 5-1 and Table 5-2 that the average time per DPC is very close to the minimum time. The maximum time, however, is almost double the minimum time. One reason for this is that all DPC's within the entire system are queued into one FIFO queue. This means that the PCI10FR DPC's are inserted in the queue along with the video card's DPC's, the network card's DPC's, the keyboard's DPC's, etc.. There is also no way to prioritize these DPC's or to insert a DPC is a particular location within the DPC queue. Only two commands are provided by Microsoft: insert a DPC at the rear of the queue, or delete a DPC from the queue.

Another reason for the occasional doubling of time required to service a DPC is that Windows NT, like many operating systems, performs garbage collection on a periodic basis. This means that Windows NT can and will stop everything it is doing to free up memory. In addition, the Windows NT Virtual Memory Manager is constantly busy paging memory to and from the hard drive. Not only does this require interrupts and DPC's, but it also causes the hard drive head to move back and forth from the paged memory file to the file being used to archive the telemetry data being transferred to host memory. There is nothing that can be done about garbage collection or the paging of memory. However, one can store telemetry data on a different hard drive than the one being used by Windows NT for paging.

Section 6. Testing and Debugging

6.1 Test Description

The PCI10FR Prototype Board was designed as a prototype board to:

a. Test the new PIFS Chip.

b. Test the V962PBC.

c. Test the use of PCI bus technology.

d. Test the performance of Windows NT as a soft real-time operating system.

e. Apply lessons learned to the development of the RLPC and other Desktop Satellite Data Processor (DSDP) PCI boards.

Since the PCI10FR Prototype Board is a prototype, formal test plans and procedures were never prepared. However, the PCI10FR Debugger application was developed to test all of the features of the board. A brief user's guide for the PCI10FR Debugger is provided in Appendix A.

6.2 Test Elements

The following list describes the hardware and software necessary to test the PCI10FR Prototype Board:

a. A high-end computer with a Pentium or Pentium-Pro microprocessor, a PCI local bus with at least one free slot, 32-Mbytes of RAM, a 2-Gbyte fast hard drive (an ultra-wide SCSI drive is preferred), a video monitor, a keyboard, and a mouse.

b. Microsoft Windows NT 3.51 should be installed on the computer.

c. A PCI10FR Prototype Board should be installed in one of the free PCI slots.

d. A source of CCSDS and weather data should be available, along with the necessary hardware to output this data via RS-422 at various data rates up to 15 Mbps.

e. A logic analyzer is highly desirable for measuring ISR and DPC latencies.

f. The PCI10FR Device Driver should be installed in the "Drivers" directory, registered with the Windows NT Registry and loaded. See Sections 4.3 and 4.4 of this document for information concerning building the PCI10FR Device Driver, registering it, and loading it.

g. The PCI10FR Debugger application should be installed.

6.3 Results

The PCI10FR Prototype Board was thoroughly and successfully tested with CCSDS data using the PCI10FR Debugger application. Test data was processed by the board, the data was transferred to host memory via DMA, and the data was archived to a hard drive. A data rate of 15 Mbps on the Dell OptiPlex GXpro Dual CPU 200 MHz computer was achieved with no errors in the data. Beyond this rate, the return link data FIFO's started overflowing.

Very little testing was performed with weather data, and no structured testing was done with the forward link command function.

Appendix A.
PCI10FR Debugger User’s Guide

Appendix A.PCI10FR Debugger User's Guide

A.1 Introduction

The PCI10FR Debugger application was written to fully debug all aspects of the PCI10FR Prototype Board. This section provides a brief user's guide for the program.

```
******************************************************************************
******************************************************************************
***************************** PCI10FR DEBUGGER *******************************
********************************* MAIN MENU **********************************
**                                                                         ***
** (0)  Board Status                        (C) PIFS Status                ***
**                                                                         ***
** (1)  Reset All Status                    (D) PIFS Reg Dump              ***
**                                                                         ***
** (2)  Save Configuration to File          (E) ReedSolomon Status         ***
**                                                                         ***
** (3)  Load Configuration                  (F) ReedSolomon Reg Dump       ***
**                                                                         ***
** (4)  ResetBoard/Test Registers           (G) Display Fifo Buffer X      ***
**                                                                         ***
** (5)  Rate Test                           (H) Dump all Fifo's to Memory  ***
**                                                                         ***
** (6)  Display Config Space                (I) Set Fwd Link FIFO Freq.    ***
**                                                                         ***
** (7)  Test Data Channels                  (J) View/Set FIFO Flags        ***
**                                                                         ***
** (8)  Forward Link Loopback Test          (K) Test Block Read/Write      ***
**                                                                         ***
** (9)  Set PIFS for Weather                (L) Set Routing Addr.          ***
**                                                                         ***
** (A)  Read RS routing table RAM           (M) Reset All Fifo's           ***
**                                                                         ***
** (B)  Load Routing Table Ram              (S) Scan & Dump Buffers        ***
**                                                                         ***
** (U)  File Scan Routines                  (P) Interrupt Test             ***
**                                                                         ***
** (Y)  Dump Buffer to File                 (T) QuickScan Buffer for Sync's***
**                                                                         ***
** (Z)  Toggle Debug Flag                   (V) Data Status                ***
**                                                                         ***
** (^)  Quit(Save Config in config.cnf)     (&) Quit W/O Saving Config     ***
**                                                                         ***
******************************************************************************
******************************************************************************
******************************************************************************

Select next command:
```

A.2 Main Menu

Figure A-1. Main Menu

The Main Debug Menu is displayed when the DEBUG program is started. The displays generated throughout this section show the results of capturing a CCSDS-simulated data set. The data set contained 128 232-byte Frames with Reed-Solomon Encoded data (Interleave 1) with a frame sync of $1ACFFC1D. The data set was sent 400 times so that 51200 frames were input to the card. The data set contained seven different Virtual Channel Identifier’s (VCID’s) (0-5,7). Each frame also contained packet data, but this is beyond the scope of this document.

Throughout this section, anytime a number is preceded by an $, this means it is a Hexadecimal Number.

All bit field selections will toggle that bit and redisplay the menu. Multiple bit fields will prompt for either Hexadecimal or Decimal values.

Some menu entries require you to type a 99 to exit. If you accidentally type an ASCII character, the display may start to roll. If it does, then you must type <CTRL>-C to exit the program and then restart it.

A.3 Board Status Menu

```
Main Control Register = f7fc8006
 Status Register  0    = fffff
 Status Register  1    = fffff
 Status Register  2    = f00001c3
 Programmable Flag Reg = ffffffff
 Misc Control Reg      = ffff0200

                                         BOARD STATUS

         Julian day = 524                                 Mar 18,1997 08:25:53

           PIFS Chip  ( CCSDS )                         Reed Solomon ( PassThrough )


         Lock Frames          51200                      Input Frames         51200
         FlyWheel Frames      3                          Long Frames          0
         BackToSearch Frames  1                          Short Frames         0
         CRC Errors           0

         Search Mode
         Input Data Rate      12.4Mbps

         Channel          MC Count              MC Seq Errors

         1                  51200                    399
         2                      0                      0
         3                      0                      0
         4                      0                      0

        VCID      Channel_1          Channel_2          Channel_3         Channel_4

                Count  SeqErr       Count  SeqErr       Count  SeqErr       Count  SeqErr

         0      1600   399          0      0            0      0            0      0
         1      1600   399          0      0            0      0            0      0
         2      12000  399          0      0            0      0            0      0
         3      12000  399          0      0            0      0            0      0
         4      12000  399          0      0            0      0            0      0
         5      12800  399          0      0            0      0            0      0
         6      0      0            0      0            0      0            0      0
         7      1600   399          0      0            0      0            0      0


Total VCID's           =    51200
Total Vcid Seq Errors  =     2793

Enter [Q]uit, Any other to Board Menu
```

Figure A-2. Board Status Menu

Entering a 0 from the main menu gives you a display of the board's complete status. This display updates every few seconds.

The Board registers are displayed at the top, then the PIFS and Reed-Solomon Status are displayed, the master channel sequence and VCID counts are displayed, and finally the Total VCID's and VCID errors are displayed. Only if a channel is opened will the master channel and VCID counts be displayed.

Entering a Q will return you to the Main menu. Any other character will bring up the Board Menu.

A.4 Board Menu

```
Board Menu (Any Character except Q from Board Status Display)

 Main Control Register = f7fc8006
 Status Register  0    = ffff0
 Status Register  1    = ffff0
 Status Register  2    = f00001c3
 Programmable Flag Reg = ffffffff
 Misc Control Reg      = ffff0200

FIFO #          Empty           Almost Empty            Almost Full             Full
   1             ***
   2                                                                            ***
   3                                                                            ***
   4                                                                            ***



[0] ...Reset PIFS Chip                          [1] ...Reset RS Chip
[2] ...Reset PIFS/RS Fifo                       [3] ...Reset FIFO Stackers
[4] ...Stackmode (Pad )                         [5] ...Stackorder  ( lluu )
[6] ...Reset FIFO # 1                           [7] ...Reset FIFO # 2
[8] ...Reset FIFO # 3                           [9] ...Reset FIFO # 4
[A] ...TXSEL ( 422Out = FLOut )                 [B] ...LEDBIT -*-
[C] ...E/D Fifo 1 Int. ( Enabled  )             [D] ...E/D Fifo 2 Int. ( Disabled )
[E] ...E/D Fifo 3 Int. ( Disabled )             [F] ...E/D Fifo 4 Int. ( Disabled )
[G] ...NCO Clk ( FlTestClk )                    [H] ...Reset FwdLink Fifo
[I] ...Reset FL P/S Converter                   [J] ...
[R] ...Reset PIFS/RS Status                     [Q] ... Quit


Enter Selection...
```

Figure A-3. Board Menu

This display shows the Board register values and the FIFO Flag Status. It also gives you the ability to Reset the Chips(PIFS,RS), Reset Chip Status(PIFS,RS), Reset all FIFOs, Enable/Disable interrupts, turn on/off the Light Emitting Diode (LED), and set Stacker order and mode.

Entering a Q will return you to the Board Status Menu.

A.5 PIFS CCSDS Status Menu

```
PIFS CCSDS Status Menu (C)


******************************************************************
                Parallel Integrated Frame Sync(PIFS)

                          STATUS REGISTERS
Register[00]...c00000e7                 Register[01]...c0006001
Register[02]...00000003                 Register[03]...98dd019d
Register[04]...00000000                 Register[05]...00000000
Register[06]...00000000                 Register[07]...0000c800
Register[08]...00000000                 Register[09]...0000c800
Register[10]...00000003                 Register[11]...00000001
Register[12]...00000000                 Register[13]...00000000
Register[14]...00000000                 Register[15]...00000000
Register[16]...00000000                 Register[17]...00000000

                          SETUP REGISTERS
Register[18]...0804c2c0                 Register[19]...1acffc1d
Register[20]...00000000                 Register[21]...ffffffff
Register[22]...00000000                 Register[23]...20000003
Register[24]...000000e7                 Register[25]...041400e7
Register[26]...00000000                 Register[27]...00000000
Register[28]...0000375a                 Register[29]...019b3d58
Register[30]...00030450                 Register[31]...00000000

Julian Day   0                          00:03:55

                                                Setup Register[18]...0804c2c0
Check Frames            0                       FlyWheel Mode
Lock Frames             51200                   CCSDS
Flywheel Frames         3                       Chip Enabled
Back To Search          1                       Serial Input
CRC Errors              0                       Serial Port [0] Input
Inverted Frames         0                       Timeout.. ~31msec
Sync Errors             0                       NRZ-L Input
Slip Errors             0                       Input Data Rate 12.38Mbps
Sync Errors FrameSync   0                       FrameLength..232
Slip Errors FrameSync   0                       SyncPattern...1acffc1d
Output Mode  Word                               EOF Pin Enabled
Pwenout0.. Lock Frames


[R]eset Status Registers                        [T]oggle Chipgo
[P]arameter Change CCSDS                        [S]et Pifs Time
[Q]uit, Return to Main Menu                     [U]pdate Status Page
```

Figure A-4. PIFS CCSDS Status Menu

Entering a C from the main menu gives you a display of the PIFS CCSDS Status (if Chipmode-Register 18 bit 27 is set to a 0) or the PIFS Weather Status (if Chipmode-Register 18 bit 27 is set to a 1) Menu.

All the Status Registers($0-$17) and Setup Registers($18-$31) are displayed at the top with status and setup information displayed next.

Entering:	U	Updates the status counts.
	R	Resets the status counts to zero.

S	Loads Time into PIFS Chip.
T	Toggles the PIFS Chip Enable bit.
P	Takes you into the Parameter Change Menu.
Q	Quit and return to the Main Menu.

A.6 PIFS CCSDS Parameter Menu

```
PIFS CCSDS Parameter Menu (P from Pifs Status menu)



                PIFS CCSDS Parameters

[0].. CHIPGO            1               [1].. CHIPMODE          0
[2].. RESYNC            0               [3].. RSSTATUS          0
[4].. INTRST            0               [5].. SYNCERRST         0
[6].. ASYNCBYPASS       0               [7].. INPUTSEL          1
[8].. SERIALSEL         0               [9].. TIMEOUTEN         1
[A].. TIMEOUTSEL        4               [B].. NRZSEL            0
[C].. RATESEL           2               [D].. RATEGO            1
[E].. SENSE             1               [F].. PERIODEN          0
[G].. PERIODMODE        0               [H].. PERIODCLR         0
[I].. PERIODRATE        0               [J].. SYNCSIZE          32
[K].. UPPERSYNCMARK     1acffc1d        [L].. LOWERSYNCMARK     0
[M].. UPPERSYNCMASK     ffffffff        [N].. LOWERSYNCMASK     0
[O].. FLYWHEELTOL       3               [P].. CHECKTOL          0
[R].. ERRTOL0           0               [S].. SLIPTOL0          0
[T].. ERRTOL1           0               [U].. SLIPTOL1          0
[V].. FRMLEN0           231             [W].. FRMLEN1           0
[X].. CRCEND            231             [Y].. CRCOFFSET         4
[Z].. CRCSETCLR         1               [!].. CRCEN             0
[@].. BESTMATCHEN       0               [#].. BTDEN             0
[$].. BTDOFFSET         4               [].. RUNMODE            0
[^].. DAYCOUNTER        14170           [&].. PFIELD            0
[*].. PFIELDEN          0               [(].. INTEXTSEL         0
[)].. TIMELOAD          0               [_].. MSOFDAY           26951000
[+].. STATUSLEN         0               [|].. STATUSEN          0
[>].. EOFEN             1               [=].. SOFEN             0
[-].. OUTMODE           1               [<].. TRAILERORDER      0
[;].. SCLFMASK0         4               [{].. SCLFMASK1         0
[}].. TIMELEN           3               [:].. TIMEEN            0
[.].. USOFMS            0

Enter Selection to Change
```

Figure A-5. PIFS CCSDS Parameter Menu

The PIFS-CCSDS Parameter Menu allows you to completely configure the PIFS chip for CCSDS processing. Refer to the PIFS hardware document for explanations of each field.

Entering a Q will return you to the Main Menu.

A.7 PIFS Weather Status Menu

```
PIFS Weather Status Menu (C from main Menu & CHIPMODE = 1)


******************************************************************
                Parallel Integrated Frame Sync(PIFS)

                          STATUS REGISTERS
Register[00]...00000000                Register[01]...e8000000
Register[02]...00000007                Register[03]...ff87023a
Register[04]...00000000                Register[05]...00000000
Register[06]...00000000                Register[07]...00000000
Register[08]...00000000                Register[09]...00000000
Register[10]...00000000                Register[11]...00000000
Register[12]...00000000                Register[13]...00000000
Register[14]...00000000                Register[15]...00000000
Register[16]...00000000                Register[17]...00000000

                          SETUP REGISTERS
Register[18]...04000000                Register[19]...00000000
Register[20]...00000000                Register[21]...00000000
Register[22]...00000000                Register[23]...00000000
Register[24]...00000000                Register[25]...00000000
Register[26]...00000000                Register[27]...00000000
Register[28]...00000208                Register[29]...036b36b0
Register[30]...00170000                Register[31]...00000000

WEATHER Data                           Chip Disabled
Parallel Input Port [0]                No Input Data
Timeout Disabled
Output Mode Byte                       EOF Pin Disabled
Pwenout0.. no Frames

****************          OPTIONS     **********************

Reset [S]tatus Registers               Toggle [E]nd of Frame Enable

[T]oggle Chipgo                        Parameter [C]hange

[P]WENOUT0 Change                      [U]pdate Status Page

[Q]uit,Return to Main Menu             [R]egister Change
```

Figure A-6. PIFS Weather Status Menu

The PIFS Weather Status Menu is similar to the CCSDS menu in that it displays the Status and Setup registers, some setup information, and then an Options area. The options are the same as the CCSDS menu, although the character entered may be different.

This menu is only entered if the Chipmode (Register18 Bit 27) is set to a 1.

A.8 PIFS Weather Parameters Menu

```
PIFS Weather Parameters (C from PIFS Status & CHIPMODE = 1)


                        PIFS Weather Parameters
                       Reg_18   04000080
[0].. CHIPGO       0            [1].. CHIPMODE          1
[2].. RESYNC       0            [3].. RSSTATUS          0
[4].. INTRST       0            [5].. SYNCERRST         0
[6].. ASYNCBYPASS  0            [7].. INPUTSEL          0
[8].. SERIALSEL    0            [9].. TIMEOUTEN         0
[10].. TIMEOUTSEL  0            [11].. NRZSEL           0
[12].. RATESEL     0            [13].. RATEGO           1
[14].. SENSE       0            [15].. PERIODEN         0
[16].. PERIODMODE  0            [17].. PERIODCLR        0
[18].. PERIODRATE  0
                       Reg_19   00000000
[19].. PNSET       0            [20].. PNENB            0
[21].. PNDECOD     0            [22].. PNENBI           0
[23].. PNENAI      0            [24].. PNSHIFTB         0
[25].. PNSHIFTA    0            [26].. STWDSET          0
[27].. SYNC24EN    0            [28].. WDCOMP           0
[29].. PACKDATA    0            [30].. MSB              0
[31].. FLFRMHD     0            [32].. REDUNC           0
[33].. WSNIBBL     0            [34].. WSFRMHD          0
[35].. WSFLNTH     0            [36].. FILBITEN         0
[37].. CRCSET      0            [38].. CRCENA           0
[39].. CRCENB      0            [40].. BLOCKCT          0
[41].. ODDEVEN     0            [42].. SUBLKCT          0
[43].. FLFIELD     0
                       Reg_20   00000000
[44].. WDSZB       0            [45].. WDSZA            0
[46].. HEADSZ      0            [47].. FILBITS          0
                       Reg_21   00000000
[48].. POLYB       0            [49].. POLYA            0
                       Reg_22   00000000
[50].. BLKSZB      0            [51].. BLKSZA           0
                       Reg_23   00000000
[52].. SYNCLNTH    0            [53].. CRCPOLY          0
                       Reg_24   00000000
[54].. WSFLOFF     0            [55].. SYNCWRDB         0
                       Reg_25   00000000
[56].. FLFOFF      0            [57].. SYNCWRDA         0
          Reg_ 26  00000000                             Register 27  0
[58].. MASKB       0            [59].. MASKA            0

Enter Selection to Change  or 99 to Quit
```

Figure A-7. PIFS Weather Parameter Menu

The PIFS Weather Parameter Menu allows you to completely configure the PIFS chip for Weather Data processing. Refer to the PIFS hardware document for each field setting.

A.9 PIFS Register Dump

```
PIFS Register Dump ( D from Main Menu)



Reg[00] c00000e7                        Reg[18] 0804c2c0
Reg[01] c0006000                        Reg[19] 1acffc1d
Reg[02] 00000009                        Reg[20] 00000000
Reg[03] f37c015a                        Reg[21] ffffffff
Reg[04] 00000000                        Reg[22] 00000000
Reg[05] 00000000                        Reg[23] 20000003
Reg[06] 00000000                        Reg[24] 000000e7
Reg[07] 0000c800                        Reg[25] 041400e7
Reg[08] 00000000                        Reg[26] 00000000
Reg[09] 0000c800                        Reg[27] 00000000
Reg[10] 00000003                        Reg[28] 0000375a
Reg[11] 00000001                        Reg[29] 019b3d58
Reg[12] 00000000                        Reg[30] 00030450
Reg[13] 00000000                        Reg[31] 00000000
Reg[14] 00000000
Reg[15] 00000000
Reg[16] 00000000
Reg[17] 00000000

Enter Register # to Write or 99 to Exit
```

Figure A-8. PIFS Register Dump Menu

The PIFS Register Dump Menu displays the PIFS Status Registers ($0-$17) and the PIFS Setup Registers ($18-$31). Only registers $18-$31 are writeable.

Enter the Register # then the Hexadecimal value to be written. It will keep prompting you until you enter a 99 to return to the main menu.

A.10 Reed-Solomon Status Display

```
Reed Solomon Status Display (E from Main Menu)



                                        REED-SOLOMON Status Routine


 Mode..PassThrough

         Chip Enabled
         Input Frames          51200                    Frame Length          232
         Long Frames           0
         Short Frames          0                        Timeout 1/2 Sec
                                                        Quality Annotation OFF



[E]nable/Disable RS Chip              [P]arameter Change
[R]eset Status Registers
[S]et Reed-Solomon Mode
[C]hange Frame Length/CodeWord        ReedSolomon Register Du[m]p
[U]pdate Status Page                  [I]nterleave change
[T]oggle Quality Annotation           [Q]uit, Return to Main Menu
```

Figure A-9. Reed-Solomon Status Display

The display shows the Reed-Solomon statistics and some setup parameters.

Entering:	E	Toggles the chip enable bit.
	R	Resets the statistics to zero.
	S	Allows you to set the Reed-Solomon Error Detection/Correction Mode.
	C	Changes the Frame and Codeword Size of the incoming Data.
	U	Updates the statistics.
	T	Toggles the Quality Annotation Bit (Register 26 Bit 12).
	P	Changes any Reed-Solomon Parameter (Reed-Solomon Parameter menu)
	M	Reed-Solomon Register Dump/Change.
	I	Changes Interleave.

A.11 Reed-Solomon Parameter Menu

```
Reed Solomon parameter menu ( P from RS Status Memu)



                        Reed Solomon Parameters

    Register 21
[0].. INTERLEAVE        0               [1].. ENCLRREG5_20      0
[2].. ENPEROUTQA        0               [3].. INTIMEOUTEN       0
[4].. CONV_DB           0               [5].. DIRACCEN          0
[6].. OUTMODE           0               [7].. INPMODE           0
[8].. ERRDET106EN       0               [9].. ERRDET255EN       0
[10].. UPMODEEN         0               [11].. UPMODEPROC       0
[12].. CHIPEN           0
    Register 22
[13].. TIMEOUTSEL       0
[14].. ENCDATAOFF       0               [15].. CWLNTH           0
    Register 23
[16].. OUTMASK          0               [17].. FRMLNTH          0
    Register 25
[18].. RSTSTATUS        0               [19].. EOFEN            0
[20].. SOFEN            0               [21].. ROUTFUNCTEN      0
[22].. ROUTBASEADD      0
    Register 26
[23].. MASKOFFQA        0               [24].. INTRESET         0
[25].. SEL255STSINFO    0               [26].. ASSBOTHCS        0
[27].. NOOUTPROC        0               [28].. EN106ERRCORR     0
[29].. EN255ERRCORR     0               [30].. SELPERIODRATE    0
    Register 27
[31].. REJUNROUTFRM     0               [32].. REJUNCORRFRM     0
[33].. REJSHORTFRM      0               [34].. REJLONGFRM       0
[35].. BASEADDRQA       0
    Register 28-31
[36].. STATANNOTREG1    0               [37].. STATANNOTREG2    0
[38].. STATANNOTREG3    0               [39].. STATANNOTREG4    0
     Register 32-39
[40].. OUTLNTHREG1      0               [41].. OUTADDRREG1      0
[42].. OUTLNTHREG2      0               [43].. OUTADDRREG2      0
[44].. OUTLNTHREG3      0               [45].. OUTADDRREG3      0
[46].. OUTLNTHREG4      0               [47].. OUTADDRREG4      0
[99] Quit Routine

Enter Selection
```

Figure A-10. Reed-Solomon Parameter Menu

The Reed-Solomon Parameter Menu allows you to completely configure the Reed-Solomon chip for CCSDS processing. Refer to the Reed-Solomon hardware document for explanations of each field.

Entering a Q will return you to the Main Menu.

A.12 Current Reed-Solomon Mode Menu

```
Current Reed Solomon Mode (S from RS Menu)



CurrentReed-Solomon Mode..PassThrough

Enter New Reed Solomon Mode
 [0] Pass Through
 [1] Enable(255,223) Detection
 [2] Enable(255,223) Detection & Correction
 [3] Enable(10,6) Detection
 [4] Enable(10,6) Detection & Correction
 [5] Enable Both Detection & Correction
```

Figure A-11. Current Reed-Solomon Mode Menu

The Current Reed-Solomon Mode Menu shows you the mode of the Reed-Solomon chip and allows you to change modes. Refer to the Reed-Solomon hardware reference document for explanations of each mode.

A.13 Reed-Solomon Register Dump

```
Reed Solomon Register Dump (E from Main Menu,or R from RS Menu)



Reg[00] ffff0010                   Reg[21] ffff1321
Reg[01] ffff0000                   Reg[22] ffffa4e3
Reg[02] ffff0000                   Reg[23] ffffa0e7
Reg[03] ffff0000                   Reg[24] ffff0000
Reg[04] ffff0000                   Reg[25] ffff5004
Reg[05] ffff0000                   Reg[26] ffff1000
Reg[06] ffffc800                   Reg[27] ffff00e8
Reg[07] ffff0000                   Reg[28] ffff0000
Reg[08] ffff0000                   Reg[29] ffff0000
Reg[09] ffff0000                   Reg[30] ffff0000
Reg[10] ffff0000                   Reg[31] ffff0000
Reg[11] ffff0000                   Reg[32] ffff00e7
Reg[12] ffff0000                   Reg[33] ffff0000
Reg[13] ffff0000                   Reg[34] ffff0000
Reg[14] ffff0000                   Reg[35] ffff0000
Reg[15] ffff0000                   Reg[36] ffff0000
Reg[16] ffff0000                   Reg[37] ffff0000
Reg[17] ffff0000                   Reg[38] ffff0000
Reg[18] ffff0000                   Reg[39] ffff0000
Reg[19] ffff0000
Reg[20] ffff0000

Enter Register # to Write or 99 to Exit
```

Figure A-12. Reed-Solomon Register Dump

The Reed-Solomon Register Dump Menu displays the Reed-Solomon Status Registers ($0-$20) and the Reed-Solomon Setup Registers ($21-$39). Only registers $21-$39 are writeable.

Enter the Register # then the Hexadecimal value to be written. It will keep prompting you until you enter a 99 to return to the main menu.

A.14 Reset Board/Test Registers Menu

```
ResetBoard/Test registers (4 From Main Menu)


Do you wish to Reset Board(R) or Test Registers(T)...


                                 PIFS Reset Test
 No Pifs Reset Errors




                                 PIFS Write Register Test(all 1's)
 No Pifs Register Write 1's Errors




                                 PIFS Write Register Test(all 0's)
 No Pifs Register Write  0's Errors




                                 ReedSolomon  Reset Test
 No ReedSolomon  Reset Errors


                          ReedSolomon Write Register Test(all 1's)
 No Reed Solomon Register Write 1's Errors




                            ReedSolomon Write Register Test(all 0's)
 No Reed Solomon Register Write 0's Errors
```

Figure A-13. Reset Board/Test Registers Menu

Entering a 4 from the main menu brings you into the Reset Board/Test Registers Menu. This allows you to either Reset the entire board or test the registers.

If R is entered, you will completely reset the board (All chips, FIFO’s, and configuration).

If a T is entered, the program reads and writes the PIFS and RS registers and displays the results. You may retest the PIFS or RS or Both by entering P, R, or B.

If you Reset or Test, you must reload the configuration of the board, option 3 from main menu.

Entering a Q will return you to the main menu.

A.15 Display FIFO Buffer Menu

```
Enter Which Fifo Buffer to Display(1-4)  ?

                Fifo 2 Buffer Dump, contains 4096(1000Hex)Longwords

000000 1acffc1d 09910000 18000000 c000003d aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
000008 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
000010 aaaaaaaa aaaaaaaa aaaaaaaa aaaa0001 c0000025 bbbbbbbb bbbbbbbb bbbbbbbb
000018 bbbbbbbb bbbbbbbb bbbbbbbb bbbbbbbb bbbbbbbb bbbbbbbb bbbb07ff c0000043
000020 55555555 55555555 55555555 55555555 55555555 55555555 55555555 55555555
000028 55555555 55555555 55555555 55555555 55555555 55555555 55555555 55555555
000030 5555552e deadbeef 0aae9729 f7414060 d96c4f80 36f1839d 5d534904 2280e6e1
000038 69985a6e f6933baa 1acffc1d 09930100 18000005 c000000d ffffffff ffffffff
000040 ffffffff ffff0006 c0000004 11111111 110007c0 00000f22 22222222 22222222
000048 22222222 22222200 08c00000 1a333333 33333333 33333333 33333333 33333333
000050 33333333 33333333 0009c000 00254444 44444444 44444444 44444444 44444444
000058 44444444 44444444 44444444 44444444 44444444 07ffc000 00315555 55555555
000060 55555555 55555555 55555555 55555555 55555555 55555555 55555555 55555555
000068 55555555 55555555 5555555c deadbeef 98607ee6 b56fede6 2b81cc35 1caebb05
000070 6b5fe989 e30d3c8d e8959db0 f8a3bde7 1acffc1d 09950200 1800040f c0000417
000078 66666666 66666666 66666666 66666666 66666666 66666666 66666666 66666666
000080 66666666 66666666 66666666 66666666 66666666 66666666 66666666 66666666
000088 66666666 66666666 66666666 66666666 66666666 66666666 66666666 66666666
000090 66666666 66666666 66666666 66666666 66666666 66666666 66666666 66666666
000098 66666666 66666666 66666666 66666666 66666666 66666666 66666666 66666666
0000a0 66666666 66666666 66666666 66666666 66666666 deadbeef e3531838 9a0a01db
0000a8 2852837c cc7d0a6d a3d491fb 5f4b9a30 cef3d60f ddcb5172 1acffc1d 09970300
0000b0 18000400 c0000417 abababab abababab abababab abababab abababab abababab
0000b8 abababab abababab abababab abababab abababab abababab abababab abababab
0000c0 abababab abababab abababab abababab abababab abababab abababab abababab
0000c8 abababab abababab abababab abababab abababab abababab abababab abababab
0000d0 abababab abababab abababab abababab abababab abababab abababab abababab
0000d8 abababab abababab abababab abababab abababab abababab abababab deadbeef
0000e0 b463448f 2ed4dc20 cccedaf7 ba3ae176 abd37dea d558b320 a9289c93 166a5fbc
0000e8 1acffc1d 09990400 18000404 c0000417 adadadad adadadad adadadad adadadad
0000f0 adadadad adadadad adadadad adadadad adadadad adadadad adadadad adadadad

Enter any character to continue, [Q]uit
```

Figure A-14. Display FIFO Buffer Menu

Entering a G from the main menu brings you into the Display FIFO Buffer. You must have dumped the FIFO's using the H option of the main menu before displaying the buffer. This will allow you to display 1 of the 4 data buffers.

A.16 Dump All FIFO’s To Memory Display

```
Dump All Fifo's to Memory (H from main Menu)



FiFo Dump Routine


    0 Longwords Reads From Fifo 1,  0 Frames Read
 4096 Longwords Reads From Fifo 2, 70 Frames Read
 4096 Longwords Reads From Fifo 3, 70 Frames Read
 4096 Longwords Reads From Fifo 4, 70 Frames Read
Enter any character to continue...
```

Figure A-15. Dump All FIFO’s to Memory Display

Entering a H from the main menu causes all four data FIFO’s to be dumped to their data buffers. They may than be displayed using the G option (Display data Buffer x) from the main menu.

A.17 Display PCI Configuration Space

```
Display Configuration Space ( 6 from Main Menu )



Pci Configuration Data: ------------------
  Register        Value
  --------        -----
  Vendor Id:      0x000011b0
  Device Id:      0x00000004

  Command:        0x00000107
  Status:         0x00000003

  Rev Id:         0x000000c0
  ProgIf:         0x00000002
  SubClass:       0x000000ff
  BaseClass:      0x00000000
  CacheLine:      0x000000f8
  Latency:        0x00000000
  Header Type:    0x00000000
  BIST:           0x00000001

  Base Reg[0]:    0x40000000
  Base Reg[1]:    0x00000000
  Base Reg[2]:    0x00000000
  Base Reg[3]:    0x00000000
  Base Reg[4]:    0x00000000
  Base Reg[5]:    0x00000000
  Rom Base:       0x00000000

  Interrupt Line: 0x00000043
  Interrupt Pin:  0x00000000
  Min Grant:      0x00000000
  Max Latency:    0x00000010

Press C to see the V3 Configuration Data.
```

Figure A-16. Display PCI Configuration Space

This menu option allows a user to view the current standard PCI configuration registers on the V962PBC. Pressing any key other than ‘c’ returns to the main menu. Pressing ‘c’ brings up the first of two V962PBC-specific displays, shown in the next two figures.

```
V3 Configuration Data: ------------------------
  Register          Offset              Value
  --------          ------              -----
  Pci_Map0          0x40 - 0x43         0x00000000
  Pci_Map1          0x44 - 0x47         0x00000000

  Pci_Int_Stat      0x48 - 0x4B         0x03020008
  Pci_Int_Cfg       0x4C - 0x4F         0x00000000

  Lb_Base0          0x54 - 0x57         0x50000009
  Lb_Base1          0x58 - 0x5B         0x40060000

  Lb_Map0           0x5E - 0x5F         0x00005006
  Lb_Map1           0x62 - 0x63         0x00000000

  Lb_Io_Base        0x6E - 0x6F         0x00000000
  Fifo_Cfg          0x70 - 0x71         0x00000505
  Fifo_Priority     0x72 - 0x73         0x00000000
  Fifo_Stat         0x74 - 0x75         0x0000c400
  Lb_Istat          0x76                0x00000000
  Lb_Imask          0x77                0x00000000

  System            0x78 - 0x79         0x00000066

  Lb_Cfg            0x7B                0x00000000

  Dma_Pci_Addr0     0x80 - 0x83         0x00000000
  Dma_Local_Addr0   0x84 - 0x87         0x00000000
  Dma_Length0_Lw    0x88 - 0x89         0x00000000
  Dma_Length0_Hb    0x8A                0x00000000
  Dma_Csr0          0x8B                0x00000000
  Dma_Ctlb_Adr0     0x8C - 0x8F         0x00000000

  Dma_Pci_Addr1     0x90 - 0x93         0x00000000
  Dma_Local_Addr1   0x94 - 0x97         0x00000000
  Dma_Length1_Lw    0x98 - 0x99         0x00000000
  Dma_Length1_Hb    0x9A                0x00000000
  Dma_Csr1          0x9B                0x00000000
  Dma_Ctlb_Adr1     0x9C - 0x9F         0x00000000

Press C to see the V3 Configuration Data.
```

Figure A-17. Display V962PBC-Specific PCI Configuration Registers (1 of 2)

Pressing any key other than ‘c’ will return to the main menu. Pressing a ‘c’ will display the second of two V962PBC-specific PCI configuration register screens, as shown in the next figure.

```
V3 Configuration Data: ------------------------
  Register         Offset              Value
  --------         ------              -----
  Mail_Data0       0xC0                0x00000000
  Mail_Data1       0xC1                0x00000000
  Mail_Data2       0xC2                0x00000000
  Mail_Data3       0xC3                0x00000000
  Mail_Data4       0xC4                0x00000000
  Mail_Data5       0xC5                0x00000000
  Mail_Data6       0xC6                0x00000000
  Mail_Data7       0xC7                0x00000000

  Mail_Data8       0xC8                0x00000000
  Mail_Data9       0xC9                0x00000000
  Mail_Data10      0xCA                0x00000000
  Mail_Data11      0xCB                0x00000000
  Mail_Data12      0xCC                0x00000000
  Mail_Data13      0xCD                0x00000000
  Mail_Data14      0xCE                0x00000000
  Mail_Data15      0xCF                0x00000000

  Pci_Mail_Iewr    0xD0 - 0xD1         0x00000000
  Pci_Mail_Ierd    0xD2 - 0xD3         0x00000000
  Lb_Mail_Iewr     0xD4 - 0xD5         0x00000000
  Lb_Mail_Ierd     0xD6 - 0xD7         0x00000000

  Mail_Wr_Stat     0xD8 - 0xD9         0x00000000
  Mail_Rd_Stat     0xDA - 0xDB         0x00000000
```

Figure A-18. Display V962PBC-Specific PCI Configuration Registers (2 of 2)

Pressing any key returns to the main menu.

A.18 View/Set FIFO Flags Menu

```
View/Set FIFO Flags (J from Main Menu)



                         Current FIFO Flag Settings

     FIFO Bank 1   FIFO Bank 2   FIFO Bank 3   FIFO Bank 4   FIFO Fd Lnk
     -----------   -----------   -----------   -----------   -----------
AE             0             0             0             0             0
AF             0             0             0             0             0



(1) FIFO Bank 1          (4) FIFO Bank 4
(2) FIFO Bank 2          (5) FIFO Fd Lnk
(3) FIFO Bank 3          (6) Cancel/Quit


Choose a FIFO Bank to change:
```

Figure A-19. View/Set FIFO Flags Menu

This menu allows the user to set the Programmable Flag Registers for the four data FIFOs and the Forward Link FIFO.

After entering the FIFO number, you will be prompted for the Almost Empty value then the Almost Full Value.

Entering a 6 will return you to the main menu.

A.19 Read Reed-Solomon Routing Table RAM

```
Read RS Routing Table Ram ( A from Main Menu)


Enter any character to continue, [Q]uit
002100 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002140 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002180 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
0021c0 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002200 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002240 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002280 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
0022c0 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002300 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002340 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002380 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
0023c0 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002400 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002440 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002480 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
0024c0 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002500 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002540 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002580 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
0025c0 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002600 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002640 f070f070 f0b0f0b0 f0b0f0d0 f0f0f0e0 _*_* _*_* _*_* _*_*
002680 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
0026c0 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002700 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002740 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002780 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
0027c0 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002800 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002840 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002880 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
0028c0 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
002900 f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0 ____ ____ ____ ____
Enter any character to continue, [Q]uit
```

Figure A-20. Read Reed-Solomon Routing Table RAM

Entering an A from the main menu allows you to display the RS Routing table.

A value of $F0 means that no FIFOs are selected for RS routing of the data.

A value of $70	FIFO 1 will receive data.
A value of $b0	FIFO 2 will receive data.
A value of $d0	FIFO 3 will receive data
A value of $e0	FIFO 4 will receive data.
A value of $00	all FIFOs will receive data from the RS chip.

A.20 Load Reed-Solomon Routing Table RAM

```
Load Routing Table Ram(B from main Menu)


Load Routing Table
------------------
(1) FIFO 1     (3) FIFO 3     (5) All FIFO's
(2) FIFO 2     (4) FIFO 4     (6) Cancel

Select the FIFO(s) to which all data will be routed:
```

Figure A-21. Load Reed-Solomon Routing Table RAM

Entering a B from the main menu allows you to setup the routing table to send data to different FIFOs or all FIFOs.

A.21 Set Routing Table Addresses

Entering an L from the main menu brings you into the Set Routing Address Routine. This allows the user to set up the Reed-Solomon to route data into any FIFO depending on the Version, Spacecraft ID, and VCID of each frame.

You will be prompted for the CCSDS Version (1or2), Spacecraft ID (0-1024), each VCID, and which FIFO will receive the data.

Entering a Q returns you to the main menu.

A.22 Save Configuration

After any changes have been made, the configuration of the card should be saved in a file. Selecting a 2 from the main menu prompts you for a filename. You enter a name and an extension of ".CNF" will be appended to the name (filename.CNF).

You may also save the configuration when exiting the DEBUGGER by entering ^ (<SHIFT> 6). This saves the configuration in a file called config.cnf. If you exit by entering & (<SHIFT> 7), the DEBUGGER will not save the configuration in a file.

A.23 Load Configuration

You may load any configuration file you previously saved by selecting a 3 from the main menu. You will be prompted for a configuration filename. You enter the filename WITHOUT the extension(filename). If the system finds the file, it will load it. If it does not find the file, you will get an error message and an Audible BEEP.

A.24 File Scan Menu

```
File Scan Menu (U from Main Menu)

******************************************************************************
******************************************************************************
********************************* Scan Data File Menu ************************
**                                                                         ***
**                                                                         ***
**                                                                         ***
** [R]ead File to Buffer              [Q]uickScan Buffer                   ***
**                                                                         ***
** [D]isplay Buffer                   [S]can_DumpBuffer                    ***
**                                                                         ***
** [Z]ero Buffer                      [O]pen Data File                     ***
**                                                                         ***
** [C]lose File                       scan[E]ntireFile                     ***
**                                                                         ***
** s[K]ip File Reads                  re[W]ind File                        ***
**                                                                         ***
** [P]arameter Change                 [^] Close File Return to Main Menu   ***
**                                                                         ***
** Display Con[F]iguration Space      Search For Pattern [X]               ***
**                                                                         ***
**                                                                         ***
**                                                                         ***
******************************************************************************
******************************************************************************
******************************************************************************



Select next command:  E
```

Figure A-22. File Scan Menu

Entering a U from the main menu brings you into the File Scan Menu. This menu allows you to Scan a data file for Frame Size errors and Master Channel sequence Count errors. After entering this menu, select the P option to Change Parameters.

A.25 File Scan Parameter Menu

```
Parameter values

 [1]  Frame Size = 232

 [2]  Sequence Count Start = 6

 [3]  Sequence Count Rollover = 7f

 [4]  Stop Quick Scan Display every 1000 Lines

 [5]  Change Sync Pattern 1acffc1d

 [6]  Max Errors Allowed Before Pausing ..25

Enter selection to change or [Q]uit
```

Figure A-23. File Scan Parameter Menu

With this menu you can change Frame Size [1], Frame Sync Pattern [5], Maximum Errors allowed before pause [6], and stop Quick scan display after a specified number of lines [4]. Options [2] and [3] are not implemented. After changing parameters, enter Q to return to Scan File Menu.

You may now open a data file[O]. If the file is located, it will be opened and the program will wait for your next entry. If the file cannot be found, you will get an error message and a warning Beep.

A.26 Scan Entire File Display

```
Scan Entire File ( E from Scan File menu )

 10000 Frames Read From File
 20000 Frames Read From File
 30000 Frames Read From File
 40000 Frames Read From File
 50000 Frames Read From File
 51200 Frames Read From File
                 1200 Syncs Found, Read 6

 2969600 Longwords Read with 51200 Syncs Found
Enter any character to continue...
```

Figure A-24. Scan Entire File Display

You may Scan the entire file [E]. This option reads in the file, checks for proper frame size and sequence counts. It displays the number of syncs found and any errors found.

A.27 Quick Scan Buffer Display

```
Quick Scan Buffer( Q from Scab Buffer Menu)


Scan Buffer Routine.....Enter 1 to Display ,0 noDisplay
Address                 First 24 Bytes of Frame                  Size (Bytes)    Sync#
004C8E70.. 1acffc1d  09955011  1fff6666  66666666  66666666  66666666   0 ###  51       1
004C8F58.. 1acffc1d  09975111  1fffabab  abababab  abababab  abababab   232     2
004C9040.. 1acffc1d  09995211  1fffadad  adadadad  adadadad  adadadad   232     3
004C9128.. 1acffc1d  099b5314  1fffafaf  afafafaf  afafafaf  afafafaf   232     4
004C9210.. 1acffc1d  09955412  1fff6666  66666666  66666666  66666666   232     5
004C92F8.. 1acffc1d  09975512  1fffabab  abababab  abababab  abababab   232     6
004C93E0.. 1acffc1d  09995612  1fffadad  adadadad  adadadad  adadadad   232     7
004C94C8.. 1acffc1d  099b5715  187cafaf  afafafaf  afafafaf  afafafaf   232     8
004C95B0.. 1acffc1d  09955813  187c6666  66666666  66666666  66666666   232     9
004C9698.. 1acffc1d  09975913  187cabab  abababab  abababab  abababab   232     10
004C9780.. 1acffc1d  09995a13  187cadad  adadadad  adadadad  adadadad   232     11
004C9868.. 1acffc1d  099b5b16  1fff5555  55555555  55555555  55555555   232     12
004C9950.. 1acffc1d  09955c14  1fff5555  55555555  55555555  55555555   232     13
004C9A38.. 1acffc1d  09975d14  1fff5555  55555555  55555555  55555555   232     14
004C9B20.. 1acffc1d  09995e14  1fff5555  55555555  55555555  55555555   232     15
004C9C08.. 1acffc1d  099b5f17  1fff5555  55555555  55555555  55555555   232     16
004C9CF0.. 1acffc1d  09916003  18000000  c003003d  aaaaaaaa  aaaaaaaa   232     17
004C9DD8.. 1acffc1d  09936103  18000005  c003000d  ffffffff  ffffffff   232     18
004C9EC0.. 1acffc1d  09956215  1800040f  c0030417  66666666  66666666   232     19
004C9FA8.. 1acffc1d  09976315  18000400  c0030417  abababab  abababab   232     20
004CA090.. 1acffc1d  09996415  18000404  c0030417  adadadad  adadadad   232     21
004CA178.. 1acffc1d  099b6518  18000424  c0030417  afafafaf  afafafaf   232     22
004CA260.. 1acffc1d  099b6619  1fffafaf  afafafaf  afafafaf  afafafaf   232     23
004CA348.. 1acffc1d  099f6703  1ffe5555  55555555  55555555  55555555   232     24
004CA430.. 1acffc1d  09956816  1fff6666  66666666  66666666  66666666   232     25
```

Figure A-25. Quick Scan Buffer Display

You may do a Quick Scan [Q] on the file. This checks the frame size sequence counts and displays the first 24 bytes of every frame.

You may Scan [S] the data buffer. This reads in a record, scans the record for frame syncs, and allows you to display any frame in that record.

You may Display [D] the data in the scan buffer.

You may skip [K] over frames, rewind [W] the file, or read [R] a record from the file.

Finally you may search for any 4-byte pattern in the record.

A.28 Data Status Display

```
Data Status ( V from main menu)u

 Parameter values

 [1]  Display Statistics

 [2]  Zero Status Structure

 [3]  Sort VCIDs Int32o Files

 [4]

Enter selection to change or [Q]uit   1

         Channel          MC Count              MC Seq Errors

         1                  52880                    413
         2                      0                      0
         3                      0                      0
         4                      0                      0

        VCID      Channel_1          Channel_2          Channel_3         Channel_4

                Count SeqErr       Count SeqErr       Count SeqErr       Count SeqErr

         0      1653   413         0      0           0      0           0      0
         1      1653   413         0      0           0      0           0      0
         2      11567  413         0      0           0      0           0      0
         3      11567  413         0      0           0      0           0      0
         4      11567  413         0      0           0      0           0      0
         5      13220  413         0      0           0      0           0      0
         6      0      0           0      0           0      0           0      0
         7      1653   413         0      0           0      0           0      0


Total VCID's           =    52880
Total Vcid Seq Errors  =     2891
```

Figure A-26. Data Status Display

This entry allows you to display [1] the VCID statistics for the file just received. You can zero [2] these statistics, and you can sort [3] the VCID’s into separate files.

A.29 Data Channel Menu and Display

```
Channel Information           Channel 1   Channel 2   Channel 3   Channel 4
-------------------           ---------   ---------   ---------   ---------
Priority                      High        Undefined   Undefined   Undefined
Data Type                     CCSDS Data  Undefined   Undefined   Undefined
Channel Open Status           Open        Closed      Closed      Closed
Total Data Frame Size               232           0           0           0
Padding Size                          0           0           0           0
Base Buffer Pointer           x00810020   x00000000   x00000000   x00000000
Current Event Block Pointer   x00810020   x00000000   x00000000   x00000000
Buffer Size (number of bytes)   4176000           0           0           0

ulongs Transferred per DMA         1740           0           0           0
DMA Blocks Per Event Block          150           0           0           0
Event Blocks per Buffer               4           0           0           0
Number Ulong's on Flush           98600           0           0           0
Cummulative Ulong's on Flush      98600           0           0           0
Total Frames Read                 51200           0           0           0
Extra Ulongs's read                   0           0           0           0
Number Buffer Overflows               0           0           0           0


(O)  Open data channel            (U)  Update Display
(F)  Flush all data channels      (L)  Loopback Test 232
(C)  Close all data channels      (Q)  Quit

Enter selection:
```

Figure A-27. Data Channel Menu and Display

This menu displays information about each of the four data channels, and allow a user to open a data channel (O), flush all data channels (F), and close all data channels (C). For convenience, one can press (U) to update the display. The (L) option actually inserts a 232-byte frame directly into the PIFS chip. If the rest of the board is set up correctly, this frame will generate an interrupt and will single step testing of the data channels and interrupts.

The (Q) option returns to the main menu.

Appendix B.
PCI10FR: Programming the FIFO Flags

Appendix B.PCI10FR: Programming the FIFO Flags

B.1 Introduction

Section B.2 contains an email message from Ken Winiecki on July 19, 1996 about programming the FIFO flags. Mr. Winiecki is one of the hardware designers of the PCI10FR Prototype Board, and the primary hardware designer for the new RLPC Board.

B.2 Programming the FIFO Flags

The following information pertains to setting the programmable almost-full and almost-empty flag offsets ("PAEFO" and "PAFFO") of the IDT72241 FIFO memory chips used on the PCI10FR card for stacking and routing the data output from the RSEDC chip (return-link data) and for buffering the forward-link data. These offsets default at reset to 7, meaning the PAEFs will assert when the number of filled bytes in each FIFO drops to 7, and the PAFFs will assert when the number of empty bytes in each FIFO drops to 7.

The PCI10FR has 16 byte-wide FIFOs arranged in 4 banks of 4. The stream of data bytes normally comes from the RSEDC and is latched by the bank(s) normally selected by the RSEDC. In order to provide ease of programming and testing of the FIFOs, the PCI10FR also provides a data path from the local bus to the FIFO banks and FIFO bank selection through a special board address 0x280000. Data bits 7-0 carry the data byte, and data bits 15-12 select the FIFO bank(s) to latch it; other data bits don't matter. In particular, Bank 1 is selected if bit 12 is 0, Bank 2 is selected if bit 13 is 0, Bank 3 is selected if bit 14 is 0, and Bank 4 is selected if bit 15 is 0; multiple banks may be chosen to latch the same data.

Within a bank, board logic interleaves the sequential data bytes among the 4 FIFOs to form the upper-upper-byte, upper-middle-byte, lower-middle-byte, and lower-lower-byte of a 32-bit word. Therefore, to write the same byte to all 4 FIFOs in a bank, it must be written 4 times sequentially, and to write two bytes sequentially to a single FIFO, the first byte must be written to all 4 FIFOs in the bank before the second byte can be written.

Each FIFO chip has 4 byte-wide programmable offset registers which are filled sequentially from the data stream when a special load signal is asserted. This is accomplished on a per-bank basis through the Main Board Control Register. In particular, all 4 FIFOs of bank 1 are in load-mode if bit 25 is 0, all 4 FIFOs of bank 2 are in load-mode if bit 23 is 0, all 4 FIFOs of bank 3 are in load-mode if bit 21 is 0, and all 4 FIFOs of bank 4 are in load-mode if bit 19 is 0; other bits function as specified in the documentation.

The first offset register pair forms the 12-bit PAEFO and the second pair forms the 12-bit PAFFO. In particular, bits 7-0 of byte 1 are bits 7-0 of the PAEFO, bits 3-0 of byte 2 are bits 11-8 of the PAEFO, bits 7-0 of byte 3 are bits 7-0 of the PAFFO, and bits 3-0 of byte 4 are bits 11-8 of the PAFFO; other bits don't matter. Note that since the four FIFOs within a bank are arranged with their outputs in parallel to form a 32-bit word, all PAEFOs within a bank must be programmed the same, as must all PAFFOs. Also note that the offsets describe both a number of bytes of a single FIFO as well as a number of 32-bit words of the bank (1/4 of the number of bytes in the bank).

WARNING

Any data output from the RSEDC during FIFO flag offset programming will be lost, so either the setup should occur before the start of the data session (preferrable), or the RSEDC should be suspended during the setup (hoping the PIFS-to-RSEDC FIFO can absorb the blocked data). Also, any data read from the FIFOs during FIFO flag offset programming will only be the contents of the offset registers, not data.

For example, say we want the PAEFs of FIFO banks 1 and 3 to assert when the number of filled bytes per bank drops to 256, and the PAFFs of those same banks to assert when the number of empty bytes per bank drops to 1 K. To program banks 1 and 3, bits 25 and 21 of the Main Board Control Register must be set to 0 (and the other bits set appropriately). 256 bytes in a bank is 64 (0x040) bytes in each FIFO, so PAEFO bits 7-0 are 0x40 and bits 11-8 are 0x0. Similarly, 1 KB in a bank is 256 (0x100) bytes in each FIFO, so PAFF0 bits 7-0 are 0x00 and bits 11-8 are 0x1. FIFO banks 1 and 3 correspond with data bits 12 and 14 being 0 (and bits 13 and 15 being 1), which is 0xA000. All PCI10FR board accesses are 32-bit, so the programming sequence is:

Action	Address	Data
write	0x000000	0xE5DD0000
write	0x280000	0x0000A040
write	0x280000	0x0000A040
write	0x280000	0x0000A040
write	0x280000	0x0000A040
write	0x280000	0x0000A000
write	0x280000	0x0000A000
write	0x280000	0x0000A000
write	0x280000	0x0000A000
write	0x280000	0x0000A000
write	0x280000	0x0000A000
write	0x280000	0x0000A000
write	0x280000	0x0000A000
write	0x280000	0x0000A001
write	0x280000	0x0000A001
write	0x280000	0x0000A001
write	0x280000	0x0000A001
write	0x000000	0xE7FD0000

Reading the offset registers is much simpler because it only entails setting the appropriate load-mode bits in the Main Board Control Register (just like for writing) and then performing standard FIFO-bank reads. Note that the 4 FIFOs within a bank are all read simultaneously in a 32-bit word, so only 4 reads are necessary to acquire all 16 offset registers of a FIFO bank.

For example, say we want to read the offset registers of FIFO bank 2. This means that bit 23 of the Main Board Control Register must be set to 0 (and the other bits set appropriately), and the data must be read from board address 0xA00000.

Action	Address	Data
write	0x000000	0xE77D0000
read	0xA00000	(all 4 PAEFO bits 7-0)
read	0xA00000	(all 4 PAEFO bits 11-8)
read	0xA00000	(all 4 PAFFO bits 7-0)
read	0xA00000	(all 4 PAFFO bits 11-8)
write	0x000000	0xE7FD0000

WARNING

Any data output from the RSEDC during FIFO flag offset reading will be written into the offset registers, NOT into the data FIFO, so if the RSEDC is outputting data, it MUST be suspended for the duration of the reading.

Programming the flag offsets of the forward-link FIFO is much simpler than for the return-link FIFOs because it is only one chip instead of a bank of 4. The load-mode bit in the Main Board Control Register is bit 2; the write address for offset programming data is the same as for forward-link data, 0xD00000; and the upper 24 bits of the write data don't matter. The PCI10FR does not support reading the forward-link FIFO, either data or flag offsets.

Appendix C.
CMOS SyncFIFO Specifications

Appendix D.
Dual Programmable Clock Generator Specifications

Appendix E.
Dual Programmable Clock Generator
Application Note

Appendix F. DS1620 Digital Thermometer and Thermostat Specifications

Appendix G.
DS1620 Digital Thermometer and Thermostat Application Note

Appendix H.
Atmel Serial EEPROM Specifications

Appendix I.
V962PBC Stepping Change Notifications

Abbreviations and Acronyms

AN	Application Note
API	Application Program Interface
ASIC	Application-Specific Integrated Circuit
BSS	Board Support Subsystem
CCSDS	Consultative Committee for Space Data Systems
CPU	Central Processing Unit
DCN	Documentation Change Notice
DDK	Device Driver Kit
DFD	Data Flow Diagram
DMA	Direct Memory Access
DPC	Deferred Procedure Call
DSDP	Desktop Satellite Data Processor
EDC	Error Detection and Correction
EEPROM	Electrically-Erasable Programmable Read-Only Memory
FIFO	First-In, First-Out
GOES	Geosynchronous Orbiting Earth Satellite
GSFC	Goddard Space Flight Center
HDD	Hardware Definition Document
I/O	Input/Output
IOCTL	Input/Output Control
IRP	I/O Request Packet
ISR	Interrupt Service Routine
LED	Light Emitting Diode
NASA	National Aeronautics and Space Administration
NCO	Numerically-Controlled Oscillator
NOAA	National Oceanic and Atmospheric Administration

NT	New Technology (as in Windows NT)
P2S	Parallel-to-Serial
PBC	PCI Bridge Chip
PCI	Peripheral Component Interconnect
PCI10FR	PCI 10 Mbps Frame Synchronization Reed-Solomon Error Detection and Correction board
PIFS	Parallel Integrated Frame Synchronizer
PLD	Programmable Logic Device
PN	Pseudo-Noise
RAM	Random Access Memory
RLPC	Return Link Processor Card
RS	Reed-Solomon
RSEDC	Reed-Solomon Error Detection and Correction
SCSI	Small Computer System Interface
SDD	Software Definition Document
SDK	Software Development Kit
SOMO	System Operation and Management Office
SP	Service Processor
TDM	Time Division Multiplexed
VCID	Virtual Channel Identifier’s
VLSI	Very Large Scale Integration
VME	Versa Module Eurocard